VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

INTERAKTIVNÍ WEBOVÉ APLIKACE PRO PODPORU VÝUKY
INTERACTIVE WEB APPLICATIONS SUPPORTING EDUCATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Ivan Tvorogov**
AUTHOR

VEDOUCÍ PRÁCE — **doc. Mgr. Pavel Rajmic, Ph.D.**
SUPERVISOR

BRNO 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní obor **Teleinformatika**
Ústav telekomunikací

Student: Ivan Tvorogov ***ID:*** 155253
Ročník: 3 ***Akademický rok:*** 2015/16

NÁZEV TÉMATU:

Interaktivní webové aplikace pro podporu výuky

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte problematiku vztahující se k apletům, které v navrhnete a implementujete v rámci bakalářské práce. Konkrétně jde o interaktivní podání algoritmů používaných pro neztrátovou kompresi dat: RLE, Huffmanovo kódování, aritmetické kódování, LZ77 nebo LZW. Uživatelské rozhraní a funkcionalitu implementujte pomocí HTML5 a Javascriptu.

DOPORUČENÁ LITERATURA:

[1] Beneš, B.; Sochor, J.; Felkel, P.; Žára, J.: Moderní počítačová grafika. Computer Press, Brno, 2005.

[2] Schildt, H. : Java7, výukový kurz, Computer Press, Brno, 2012.

Termín zadání: 1.2.2016 ***Termín odevzdání:*** 1.6.2016

Vedoucí práce: doc. Mgr. Pavel Rajmic, Ph.D.
Konzultant bakalářské práce:

doc. Ing. Jiří Mišurec, CSc., *předseda oborové rady*

ABSTRAKT

Tato práce se zaměřuje především na vývoj interaktivních aplikací pro vizualizaci algoritmů bezeztrátové komprese dat. Teoretická část se věnuje popisu teorií z oblasti bezztrátových kompresních algoritmů a základnímu popisu jazyků HTML5, JavaScript, CSS, které budou při vývoji použity. Praktická část je zaměřena na návrh a implementaci jednotlivých aplikací.

KLÍČOVÁ SLOVA

Bezeztrátová komprese dat, Huffmanovo kódování, LZW, RLE, Aritmetické kódování, html5, JavaScript, CSS

ABSTRACT

The project mainly aims to the development of applications for viewing lossless data compression algorithms. A theoretical part is devoted to description of theory from the area of lossless data compression algorithms and basic description of the language of HTML5, JavaScript, and CSS that will be used during the development. A practical part is focused on proposal and implementation of particular applications.

KEYWORDS

Lossless compression, Huffman coding, LZW, RLE, Arithmetic coding, html5, JavaScript,CSS

TVOROGOV, Ivan *Interaktivní webové aplikace pro podporu výuky*: bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2015/2016. 55 s. Vedoucí práce byl doc. Mgr. Pavel Rajmic, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Interaktivní webové aplikace pro podporu výuky“ jsem vypracoval(a) samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor(ka) uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil(a) autorská práva třetích osob, zejména jsem nezasáhl(a) nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom(a) následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora(-ky)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu doc. Mgr. Pavlu Rajmicovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci. Děkuji také své rodině za podporu.

Brno ....

podpis autora(-ky)

Faculty of Electrical Engineering
and Communication
Brno University of Technology
Purkynova 118, CZ-61200 Brno
Czech Republic
http://www.six.feec.vutbr.cz

PODĚKOVÁNÍ

Výzkum popsaný v této bakalářské práci byl realizován v laboratořích podpořených z projektu SIX; registrační číslo CZ.1.05/2.1.00/03.0072, operační program Výzkum a vývoj pro inovace.

Brno . . . . . .

. . . . . .
podpis autora(-ky)

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

OP Výzkum a vývoj
pro inovace

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

Interaktivní výuka v dnešní době je progresivní metoda výuky. Důležitou výhodou interaktivní výuky je názornost, která napomáhá studentům lépe porozumět probírané látce.

Tato bakalářská práce se bude věnovat tvorbě několika interaktivních aplikací, konkretně jde o implementaci bezeztrátových algoritmů komprese dat, které bude možné využít pomocí internetového prohlížeče.

V první kapitole bude vytvořen obecný přehled o webových technologiích, které pak budu používat pro tvorbu jednotlivých aplikací.

Druhá kapitola bude věnována teorii bezeztrátových kompresních algoritmů jako je Huffmanovo kódování, aritmetické kódování, RLE a LZW kódování. Budou zde popsány základní informace ohledně algoritmů a fungování těchto algoritmů.

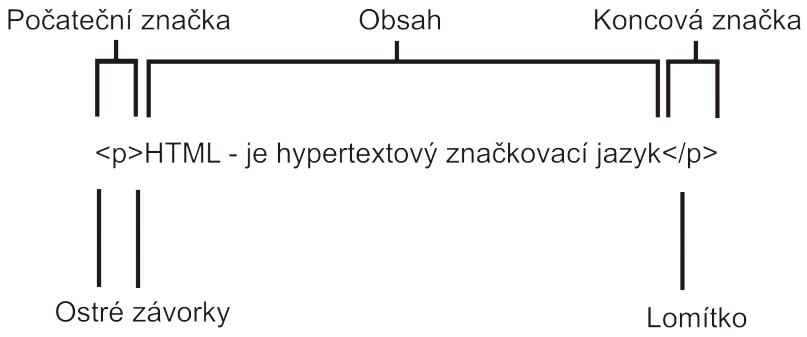
V praktické části bakalářské práce bude proveden návrh uživatelského rozhraní. Pak budou popsány implementace aplikací, vytvořené funkce a ovládání jednotlivých aplikací.

1 WEBOVÉ TECHNOLOGIE

1.1 HTML

HTML (HyperText Markup Language) – hypertextový značkovací jazyk, který popisuje strukturu a sémantiku dokumentu. Slovem hypertextový rozumíme „text, který odkazuje na jiné texty“. Jazyk HTML byl vyvinut v roce 1990 a je aplikací jazyka SGML (Standard Generalized Markup Language). SGML jazyk je komplet pravidel definující tvorbu značkovacích jazyků [2].

Jazyk HTML se skládá z elementů, které jsou reprezentovány značkami (tagy), a z jejich vlastností (atributy). Elementy můžou obsahovat také další elementy, nějaký text, a nebo být prázdné. Všechny značky a jejich vlastnosti se uzavírají mezi ostré závorky <>. Pokud značky mají jak počáteční značku (představuje název a atributy elementů uzavřené mezi znaky < a >), tak i koncovou značku (představuje lomítko a název elementů uzavřené mezi znaky < a >), říká se jim párové značky (viz obr. 1.1).

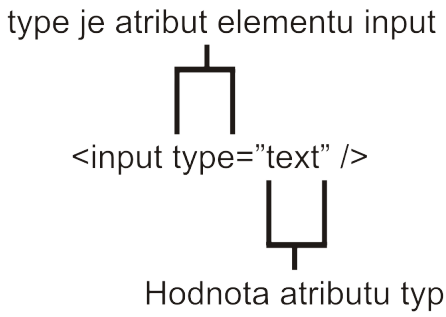


Obr. 1.1: Ukázka zápisu párových značek

Existují také prázdné elementy, reprezentované nepárovou značkou. Prázdné elementy nemají obsah a vypadají jako spojení počáteční a koncové značky, tvořené názvem elementu, atributy a jejich hodnotami, volitelnou mezerou a volitelným lomítkem [2]. K prázdným elementům patří například:

- `<input>` – element pro tvorbu formulářových prvků,
- `<br>` – řádkový zlom,
- `<img>` – vkládání obrázků do webové stránky.

Atributy a jejich hodnoty se zapisují za názvem elementu. Atribut říká prohlížeči jak má být zobrazován ten či jiný element stránky (viz obr. 1.2). Pro přidělení hodnoty atributu se používá znak =, a hodnoty atributů se uzavírají do uvozovek ("") [4].



Obr. 1.2: Ukázka zápisu atributů a jejich hodnot

1.1.1 HTML5

HTML 5 – je nejnovější verze jazyka HTML. HTML5 přináší nové prvky, které ulehčují vývoj interaktivních webových aplikací. S příchodem jazyka HTML5 by mělo dojít k nahrazování zásuvných modulů jako je Flash nebo Silverlight. Několik novinek, které přináší jazyk HTML5[2]:

- nové elementy pro strukturovaní dokumentů, například: *header*, *nav*, *footer*, které určují danou část stránky,
- velkou změnou jazyka HTML5 je schopnost pracovat s multimédii. HTML5 přináší nativní podporu přehrávání hudby a videa, a to bez použití zásuvných modulů typu Flash,
- HTML5 také přináší nové typy formulářových prvků a atributy, které umožňují omezovat a validovat vstupní data bez použití jazyka JavaScript,
- **Geolocation** – umožňuje zjišťování informace o aktuální pozici uživatele. Zjišťování aktuální pozice se provádí pomocí IP adresy anebo pomocí GPS.
- **Canvas** – pomocí tohoto elementu můžeme kreslit bitmapovou grafiku, grafy.

1.2 JavaScript

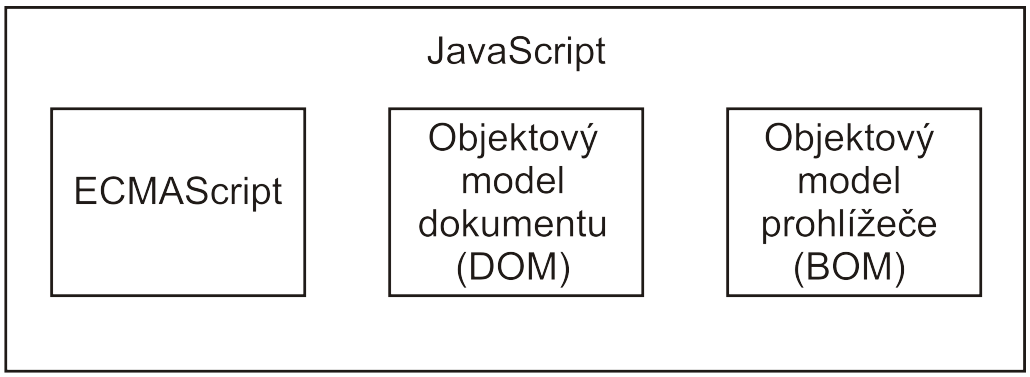
JavaScript – objektově orientovaný programovací jazyk, interpretovaný ve webovém prohlížeči. JavaScript má syntaxi podobnou jako u jazyků C, Perl a Python.

Jedná se o skriptovací jazyk, který se používá pro tvorbu interaktivních webových stránek a webových aplikací.

S JavaScriptem se můžeme setkat nejen ve webovém prohlížeči, ale také v programech jako Adobe Acrobat Reader, Adobe Photoshop, kde se využívá pro rozšíření možností těchto programů [3]. Implementace jazyka JavaScript sestavá ze tří částí (viz obr. 1.3):

- jádro (ECMAScript),
- objektový model dokumentu,

- objektový model prohlížeče.



Obr. 1.3: Implementace jazyka JavaScript

1.2.1 ECMAScript

ECMAScript je skriptovací jazyk ve standardu ECMA-262. Standard ECMA-262 definuje ECMAScript jako základ pro tvorbu dalších skriptovacích jazyků. ECMAScript je specifikace jazyka, zatímco JavaScript je jeho implementace [6].

Ve standardu ECMA-262 jsou popsané tyto části jazyka:

- příkazy,
- typy,
- syntaxe,
- klíčová slova,
- vyhrazená slova,
- operátory,
- objekty.

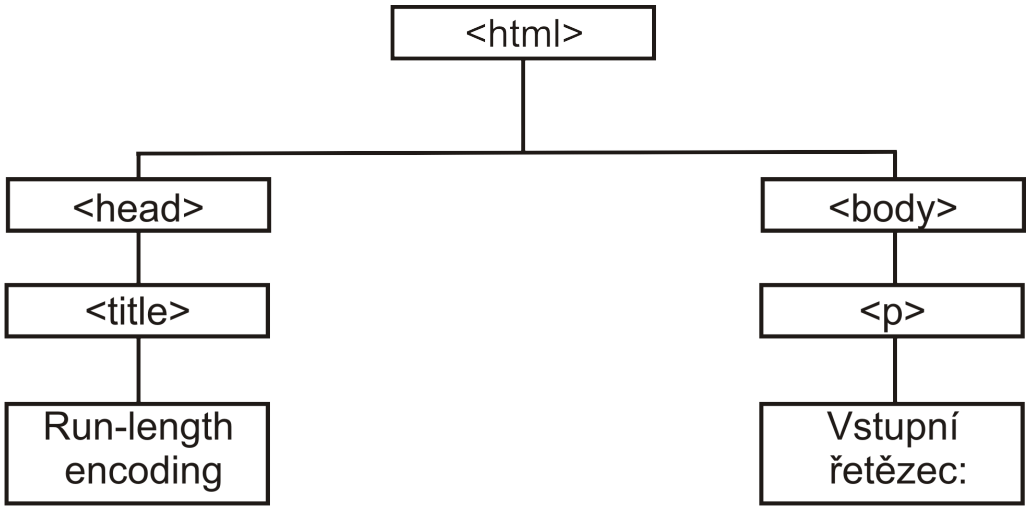
1.2.2 DOM – Document Object Model

DOM (Document Object Model) – objektový model dokumentu reprezentuje aplikační programovací rozhraní (API[1]) pro jazyky XML[2] a HTML. Objektový model dokumentu představuje dokument jako strom uzlů. Každý uzel reprezentuje různou informaci nebo značky (tagy). DOM poskytuje odstraňování, přidávání, nahrazování a upravování jednotlivých uzlů pomocí JavaScriptu nebo jiných skriptovacích jazyků [5]. Podívejme se na obrázek 1.4, který ukazuje kód HTML stránky. Tento kód si můžeme zobrazit jako strom uzlů pomocí objektového modelu dokumentu (viz obr. 1.5).

[1] Application Programming Interface
[2] Extensible Markup Language

```
<html>
<head>
   <title>Run-length encoding</title>
<head>
<body>
   <p>Vstupní řetězec:</p>
</body>
</html>
```

Obr. 1.4: Úkazka kódu HTML stránky



Obr. 1.5: Stromovitá struktura HTML dokumentu

1.2.3 BOM – Browser Object Model

BOM (Browser Object Model) – objektový model prohlížeče poskytuje práci s oknem prohlížeče. Jedná se o část implementace jazyka JavaScript, která není standardizovaná. A kvůli tomu mají různé prohlížeče svoje vlastní implementace [5].

1.2.4 Syntaxe JavaScript

V jazyku JavaScript je třeba rozlišovat velká a malá písmena, například klíčové slovo `while` musí být zapsáno jako „while“, nikoliv „While“ anebo „WHILE“. Analogicky při zapisování názvu proměnné, například „count“, „Count“ a „COUNT“ – jsou tři různé proměnné. Název proměnné musí začínat znakem $, písmenem nebo podtržítkem, další znaky můžou být čísla. V názvech proměnných nejsou povolené mezery, při zapisování dlouhých jmen můžeme namísto mezer použít podtržítko [7].

1.3 CSS

CSS (Cascading Style Sheets) – kaskádové styly umožňují úpravu vzhledu dokumentu, na rozdíl od HTML, který popisuje strukturu dokumentu. Pomocí kaskádových stylů můžeme měnit velikost textu, barvu pozadí a textu, pozici elementů. CSS se stará o to, jak má být ten či jiný element stránky vykreslen na obrazovku [4]. Kaskádové styly se dělí na následující úrovně [13]:

- CSS1 – první verze kaskádových stylů byla vydána v roce 1996. Tato verze přináší možnost změny písma, barvy pozadí a textu, zarovnání textu, obrázků a tabulek,
- CSS2 – je druhá verze kaskádových stylů, která rozšiřuje vlastnosti pro formátování písma, možnost rozestavění elementů v absolutní, relativní a fixní poloze , číslování kapitol,
- CSS2.1 – je vylepšená verze CSS2, ve které byly opravené chyby, změněny a odstraněny některé nefunkční prvky.

1.3.1 CSS3

CSS3 je poslední aktuální verze kaskádových stylů, která rozšiřuje verzi CSS2.1. Zde jsou některé možnosti, které přináší CSS3 [8]:

- **border-radius** – pomocí této vlastnosti můžeme u jednotlivých elementů nastavit zaoblené rohy. Tato vlastnost také umožňuje nastavení zaoblení pro každý roh zvlášť. Příklad zápisu: `border-radius: 20px;`,
- **opacity** – umožňuje nastavení průhlednosti u jednotlivých elementů. Příklad zápisu: `opacity: .5;`,
- **RGBA** – umožňuje definovat barvu a navíc její průhlednost. Příklad zápisu: `rgba (0,255,0,.75);`.

1.4 Vývojové prostředí

Pro vývoj jednotlivých aplikací jsem použil textový editor Sublime Text 3.

1.4.1 Sublime Text 3

Sublime Text je multiplatformní textový editor napsaný v jazyce C++. Vývojářem editoru je Jon Skinner. Funkcionalitu editoru můžeme modifikovat pomocí instalace velkého množství doplňků [12].

Sublime Text má následující výhody:

- mapa kódu,

- automatické dokončování kódu,
- zvýrazňování párových závorek,
- takzvané „Snippets“ – často používaný kód,
- zvýrazňování syntaxe pro jazyky JavaScript, HTML a CSS.

2 BEZEZTRÁTOVÁ KOMPRESE DAT

Bezeztrátová komprese – je takový typ komprese, při které nedochází ke ztrátě informace. Bezeztrátová komprese funguje takovým způsobem, že při dekompresi dostáváme stejná data, jaká jsme měli před kompresí. Bezeztrátová komprese dat se používá tam, kde potřebujeme, aby původní data a data po dekódování byla stejná, například při kompresi textů [9].K vyjádření výkonu komprese se využívá:

Kompresní poměr se definuje jako délka dat výstupních ku délka dat vstupních:

$$Kompresní\ poměr = \frac{délka\ dat\ výstupních}{délka\ dat\ vstupních} \tag{2.1}$$

Úspora – ušetření místa po komprese:

$$100 \times (1 - Kompresní\ poměr)\% \tag{2.2}$$

2.1 RLE

RLE (Run-Length Encoding) – kódování délkou běhu je jednoduchá metoda bezeztrátové komprese dat. Tato metoda spočívá ve snížení fyzické velikosti posloupnosti opakujících se po sobě znaků. Příkladem použití mohou být obrázky s velkou plochou identických barev. S touto metodou komprese dat se můžeme setkat například ve formátech obrázku jako PCX[1], TIFF[2] a TGA[3] [1].

2.1.1 Kódování

Základní varianta RLE komprese kóduje posloupnosti opakujících se po sobě znaků do dvojic (počet opakování, znak)(viz obr. 2.1).

Rozebereme si příklad z obrázku 2.1, máme vstupní posloupnost, která obsahuje 17 znaků:

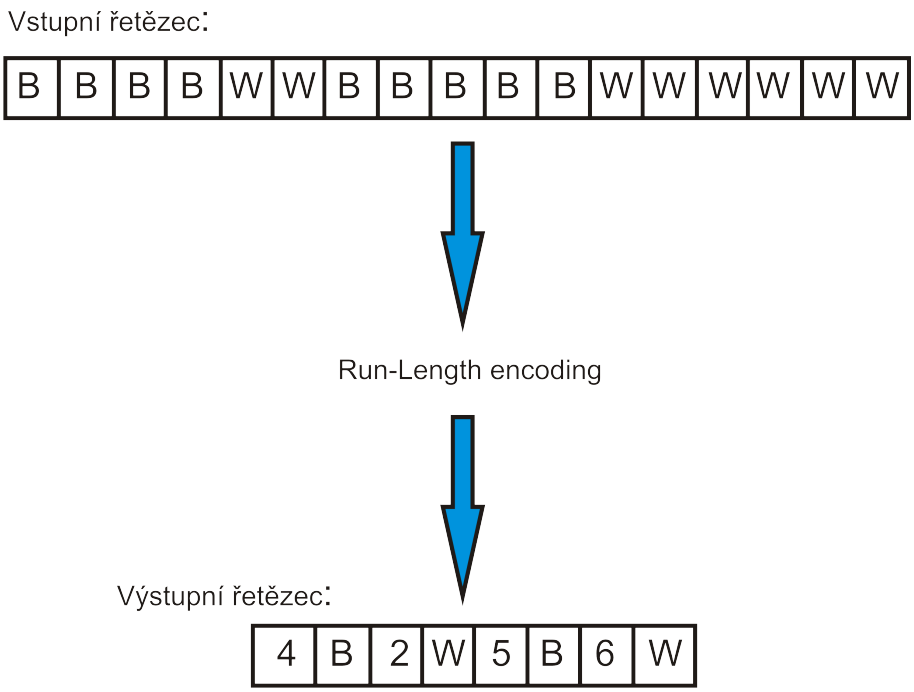
```
BBBBWWBBBBBWWWWWW
```

kde symbolem `B` označujeme černou barvu a symbolem `W` označujeme barvu bílou. Po RLE kompresi by výstupní řetězec vypadal jako posloupnost:

```
4B2W5B6W
```

Výstupní řetězec obsahuje 8 znaků, a vstupní 17. Pak můžeme spočítat kompresní poměr, který nám vyjde 2,125, což znamená, že komprimovaná data se zmenšila 2,125krát.

[1] PiCture eXchange
[2] Tag Image File Format
[3] Truevision Graphics Adapter



Obr. 2.1: Ukázka RLE kódování

Existují také případy, kdy po RLE kódování může nastat negativní komprese. Slovem negativní komprese rozumíme, že výstupní kódovaná data mají vetší objem než data před kompresí. Například máme vstupní řetězec, který obsahuje 10 znaků:

```
WBWBWBWWWW
```

pak po RLE kompresi bude posloupnost vstupního řetězce vypadat jako:

```
1W1B1W1B1W1B4W
```

což nám říká, že výstupní řetězec obsahuje o 4 znaky navíc, a tím došlo k negativní kompresi. Vzhledem k tomu je RLE komprese dat nevhodná pro posloupnosti s nízkým počtem opakujících se po sobe znaků.

2.1.2 Dekódování

Proces dekódování probíhá následujícím způsobem. Algoritmus narazí na první znak, a jestli je to číslo, pak ví, že to číslo má odpovídat počtu opakování. Pak přečte další znak, a jestli tento znak je písmeno nebo symbol, zapíše do výstupu ten znak, tolikrát, kolikrát zjistil z předchozího kroku. Proces se opakuje dokud algoritmus nenarazí na konec posloupnosti. Aby nevznikl problém s dekódováním v této bakalářské práci bylo rozhodnuto, že vstupní řetězec nesmí obsahovat čísla.

2.2 Huffmanovo kódování

David Albert Huffman vyvinul tento algoritmus v roce 1952. Huffmanovo kódování využívá takzvaného prefixového kódu. Principem prefixového kódu je, že žádné kódové slovo není předponou jiného [1].

2.2.1 Kódování

Algoritmus je založen na četnosti výskytu jednotlivých znaků v posloupnosti. Znaky, které se vyskytují nejčastěji, pak budou kódované kratší kódovou posloupností, a znaky, které se vyskytují málo budou zakódované naopak větší kódovou posloupností [10]. Algoritmus kódování probíhá ve dvou krocích:

- v prvním kroku algoritmus projde vstupním řetězcem a vytvoří tabulku četnosti výskytu jednotlivých znaků,
- v dalším kroku vytvoří binární strom a pak provede kódování vstupních dat.

Podívejme se na následující příklad, kde jako vstupní řetězec máme „boom pop“. Nejprve spočítáme četnosti výskytu jednotlivých znaků (viz tab. 2.1).

Tab. 2.1: Tabulka četnosti výskytu jednotlivých znaků

Znak	b	o	m	’ ’	p
Četnost	1	3	1	1	2

Podle četnosti výskytu znaků vytvoříme frontu, kde priorita bude podle čestnosti výskytu jednotlivých znaků (viz tab. 2.2).

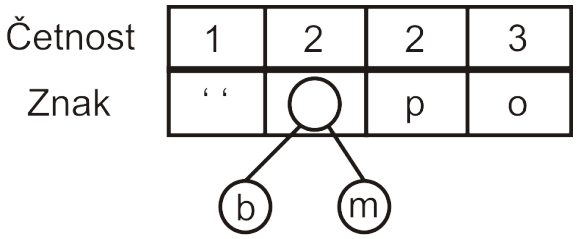
Tab. 2.2: Uspořádaní fronty pro tvorbu uzlů binárního stromu

Četnost	1	1	1	2	3
Znak	b	m	’ ’	p	o

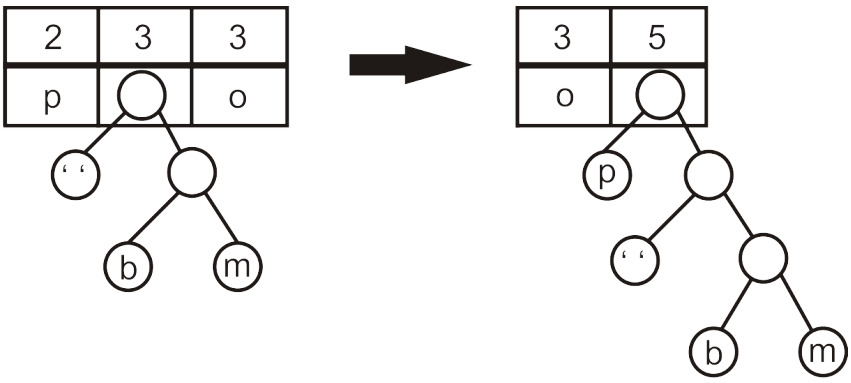
Teď’ můžeme začít tvořit uzly binárního stromu. Dva první znaky z fronty sloučíme, pak z nich vytvoříme nový uzel stromu a přidáme ho do fronty. Četnost toho nového uzlu je součet četností sečtených prvků (viz obr. 2.2).

Opakujeme předchozí kroky tak dlouho, dokud nám nezbudou dva prvky (viz obr. 2.3).

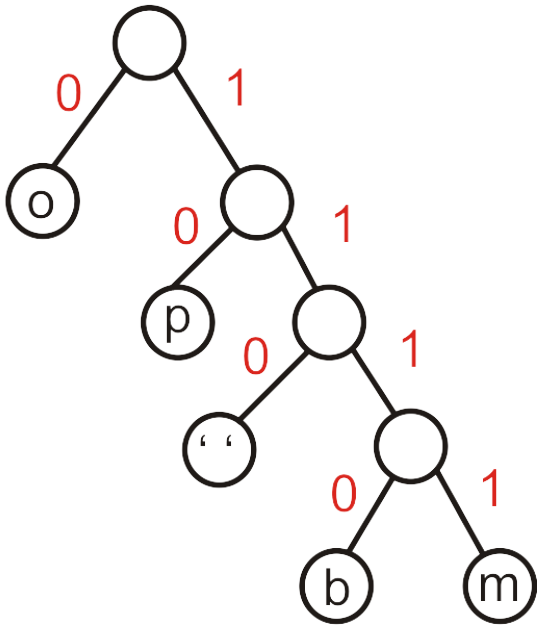
Nakonec sloučením posledních dvou prvků dostaneme výsledný binární strom a původním znakům přiřadíme binární hodnoty 1 a 0. Projdeme stromem od kořene k uzlům a budeme přidávat 0 když jdeme doleva, a 1 když jdeme doprava (viz obr. 2.4). Výsledné kódy pro jednotlivé znaky znázorňuje tab. 2.3.



Obr. 2.2: Ukázka tvorby uzlů binárního stromu



Obr. 2.3: Ukázka pokračování tvorby uzlů binárního stromu



Obr. 2.4: Ukázka výsledného binárního stromu

Tab. 2.3: Kódovací tabulka pro příklad „boom pop“

Znak	**Kod**
o	0
p	10
’ ’	110
b	1110
m	1111

2.2.2 Dekódování

Dekódování se provádí pomocí binárního stromu. Dekodér projde stromem od kořene k uzlům a takovýmto způsobem dekóduje jednotlivé znaky. Například máme zakódovaný řetězec „1110 0 1110 0“ a náš výsledný binární strom, pak dostaneme dekódovaný řetězec „bobo“.

2.3 LZW

LZW (Lempel-Ziv-Welch) – je bezeztrátový algoritmus komprese dat, který je vyvinutý Abrahamem Lempelem, Jacobem Zivem a Terry Welchem. Využívá takzvanou slovníkovou metodu, to znamená, že v průběhu kódování vytvoří slovník, který můžeme chápat jako indexovaní jednotlivých frází vstupní posloupnosti.

2.3.1 Kódování

Proces komprese dat pomocí LZW probíhá následujícím způsobem. Na začátku kódování se vytváří slovník s jednoznakovými frázemi vstupní posloupnosti, který jim přidělí indexy. Algoritmus postupně přečte každý znak vstupní posloupnosti a zkontroluje je-li ve slovníku daná fráze. Jestli daná fráze je ve slovníku, pak algoritmus čte další znak. Jestliže daná fráze ve slovníku není, pak se do výstupu zapíše index předchozí fráze, nová fráze se zapíše do slovníku a algoritmus pokračuje dál [10]. Nová fráze se tvoří z předchozí frázi a současné frázi. Teď’ rozebereme jednoduchý příklad kódování pomoci LZW. Například máme vstupní posloupnost „wabbawabba“ a původní slovník s jednoznakovými frázemi bude vypadat takto (viz tab. 2.4).

- čteme první znak „w“ a přidáme ho do původně prázdné fráze, fráze „w“ je ve slovníku,
- čteme další znak „a“ a kontrolujeme je-li fráze „wa“ ve slovníku, slovník takovou frázi nemá, přidáme novou frázi s indexem 3 do slovníku, pak do výstupu

Tab. 2.4: Původní slovník pro příklad „wabbawabba“

Slovník	**Index**
w	0
a	1
b	2

zapíšeme index předchozí fráze, což je „0“,

- čteme další znak „b“ a kontrolujeme je-li fráze „ab“ ve slovníku, „ab“ není ve slovníku, pak přidáme novou frázi do slovníku s indexem 4 a do výstupu zapíšeme „1“,
- čteme další znak „b“ a kontrolujeme je-li fráze „bb“ ve slovníku, „bb“ není ve slovníku, pak přidáme novou frázi do slovníku s indexem 5 a do výstupu zapíšeme „2“,
- čteme další znak „a“ a kontrolujeme je-li fráze „ba“ ve slovníku, „ba“ není ve slovníku, pak přidáme novou frázi do slovníku s indexem 6 a do výstupu zapíšeme „2“,
- čteme další znak „w“ a kontrolujeme je-li fráze „aw“ ve slovníku, „aw“ není ve slovníku, pak přidáme novou frázi do slovníku s indexem 7 a do výstupu zapíšeme „1“,
- čteme další znak „a“ a kontrolujeme je-li fráze „wa“ ve slovníku, „wa“ je ve slovníku, pak čteme další znak „b“ a kontrolujeme je-li fráze „wab“ ve slovníku, „wab“ není ve slovníku, pak přidáme novou frázi do slovníku s indexem 8 a do výstupu zapíšeme „3“,
- čteme další znak „b“ a kontrolujeme je-li fráze „bb“ ve slovníku, „bb“ je ve slovníku, pak čteme další znak „a“ a kontrolujeme je-li fráze „bba“ ve slovníku, „bba“ není ve slovníku, pak přidáme novou frázi do slovníku s indexem 9 a do výstupu zapíšeme „5“,
- čteme další znak „a“, to je poslední znak, pak do výstupu zapíšeme jeho index „0“.

Pak zakódovaná posloupnost bude „01220350“ a výsledný slovník bude vypadat takto (viz tab. 2.5).

2.3.2 Dekódování

Dekódování se provádí následujícím způsobem. Pro dekódování zakódované posloupnosti potřebujeme mít původní slovník s jednoznakovými frázemi. Algoritmus postupně čte indexy zakódované posloupnosti, pak do výstupu zapisuje fráze odpovídající indexu ve slovníku. Do slovníku se přidává nová fráze, která se tvoří z fráze

Tab. 2.5: Výsledný slovník pro příklad „wabbawabba“

Slovník	**Index**
w	0
a	1
b	2
wa	3
ab	4
bb	5
ba	6
aw	7
wab	8
bba	9

předchozí a plus první znak fráze současné. Slovník po kódování a slovník po dekódování jsou stejné [10].

2.4 Aritmetické kódování

2.4.1 Kódování

Aritmetické kódování provádí kódování celého vstupního řetězce do jednoho čísla. Každému znaku je přidělena odpovídající poměrná část intervalu $[0, 1)$ podle její pravděpodobnosti výskytu. V průběhu kódování je interval $[0, 1)$ postupně se zužuje na základě postupně přicházejících znaků. Každému znaků přidělí odpovídající poměrnou část z aktuálního intervalu, která pak je novým základem pro další znak. Výslednou kódovanou hodnotou je střední hodnota z posledního intervalu kódované posloupnosti. Výslednou hodnotu je třeba převést do binárního tvaru. Všechny čísla intervalu začínají 0, proto je nemusíme brát v úvahu a do binárního tvaru je nutné převést pouze desetinné části čísla. Tímto je kódování ukončeno. Výpočet potřebného počtu bitů se provádí proto, aby dekodér mohl z této binární posloupnosti jednoznačně dekódovat původní vstupní řetězec [10].

Rozebereme následující příklad. Máme vstupní řetězec „abcaab“ s pravděpodobností výskytu znaků $50\,\%$, $33,\overline{33}\,\%$ a $16,\overline{16}\,\%$. Nastavíme počáteční interval na $[0, 1)$. Pak provedeme rozdělení pravděpodobnosti výskytu znaků do intervalu $[0, 1)$ (viz tab. 2.6).
Dalším krokem bude výpočet nových hranic intervalu pro každý znak (viz tab. 2.7).

Tab. 2.6: Tabulka rozdělení pravděpodobnosti výskytu znaků pro „abcaab“

Znak	**Četnost**	**Interval**
a	0,5	[0;0,5)
b	$0,33\overline{33}$	[0,5;$0,83\overline{33}$)
c	$0,16\overline{66}$	[$0,83\overline{33}$;1)

Výpočet nových hranic provedeme pomocí těchto dvou vzorců:

$$High = Low_{old} + (High_{old} - Low_{old}) \times Range_{High} \tag{2.3}$$

$$Low = Low_{old} + (High_{old} - Low_{old}) \times Range_{Low} \tag{2.4}$$

kde $High_{old}$ – je horní hodnota intervalu,
Low_{old} – je spodní hodnota intervalu,
$Range_{High}$ – je horní hodnota intervalu kódovaného znaku,
$Range_{Low}$ – je spodní hodnota intervalu kódovaného znaku.
Příklad vypočtu nové horní hranice pro znak „a“ pomocí (2.3):

$$High = 0 + (1 - 0) \times 0,5$$

$$High = 0,5$$

a vypočet nové spodní hranice pro znak „a“ pomocí (2.4):

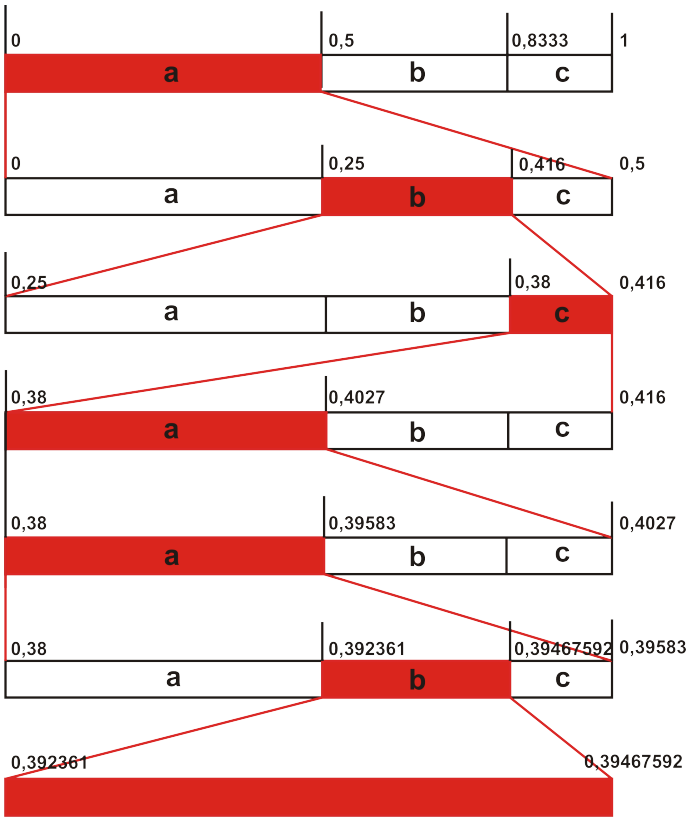
$$Low = 0 + (1 - 0) \times 0$$

$$Low = 0$$

Tab. 2.7: Tabulka vypočtu nových hranic

Krok	**Znak**	**Highold-Lowold**	**Low**	**High**
0			0	1
1	a	1	0	0,5
2	b	0,5	0,25	$0,41\overline{6}$
3	c	$0,1\overline{6}$	$0,3\overline{8}$	$0,41\overline{6}$
4	a	$0,02\overline{7}$	$0,3\overline{8}$	$0,402\overline{7}$
5	a	$0,013\overline{8}$	$0,3\overline{8}$	$0,3958\overline{3}$
6	b	$0,0069\overline{4}$	$0,39236\overline{1}$	$0,3946759\overline{2}$

V dalším kroku interval se zužuje a hodnoty intervalu Low_{old} a $High_{old}$ se mění na hodnoty intervalu pravě zakódovaného znaku.



Obr. 2.5: Ukázka kódování

Po výpočtu všech hranic dostaneme výsledný interval $[0,392361;0,3946759\overline{2})$ (viz tab. 2.5)

a výstupní číslo pak je střední hodnota z tohoto intervalu, což je $0,39351\overline{85}$. Dalším krokem je třeba výstupní číslo převést do binárního tvaru. Zaprvé spočítáme kolik bitů potřebujeme pro zakódování vstupní posloupnosti pomocí (2.5).

$$log_2\left(\frac{1}{\omega_n}\right)+1 \tag{2.5}$$

kde ω_n – je $\prod_{k=1}^{n} P(x_k)$ násobení všech pravděpodobnosti výskytu znaků v celé vstupní posloupnosti.

Příklad vypočtu potřebného počtu bitů pro zakódování vstupní posloupnosti (zaokrouhlení se provádí nahoru):

$$log_2\left(\frac{1}{0,5\times 0,33\overline{33}\times 0,16\overline{66}\times 0,5\times 0,5\times 0,33\overline{33}}\right)+1=10$$

Převedeme výstupní číslo do binárního tvaru s takovým počtem bitů, který nám vyšel v předchozím kroku, výstupní číslo je 0110010010.

2.4.2 Dekódování

Proces dekódování probíhá následujícím způsobem. Pro dekódování potřebujeme mít k dispozici pravděpodobnosti výskytu jednotlivých znaků a délku vstupního řetězce před kódováním. Délku vstupního řetězce před kódováním potřebujeme mít proto aby dekodér věděl, kdy musí proces dekódování ukončit. Výstupní číslo v binárním tvaru převedeme zpět do tvaru desítkového. Dekodér nastaví počáteční interval na $[0, 1)$ a stejné jako při kódování jej rozdělí podle pravděpodobností výskytu jednotlivých znaků. Dalším krokem zkontroluje do jakého intervalu spadá toto číslo a tím určí výsledný znak [10]. Pro výpočet nové hodnoty výstupního čísla potřebujeme následující vzorec:

$$Výstupní\ číslo = (Výstupní\ číslo - Low)/(High - Low) \tag{2.6}$$

kde $High$ – je horní hodnota intervalu, ve kterém se nachází výstupní číslo,
Low – je spodní hodnota intervalu, ve kterém se nachází výstupní číslo.

Proces dekódování pokračuje tak dlouho, dokud nebudeme mít dekódovaný celý řetězec.

Provedeme dekódování výstupního čísla 0110010010. Výstupní číslo je třeba převést zpět do desítkového tvaru, což nám vyjde 0,392578125. Je vidět že výstupní číslo spadá do intervalu [0;0,5), což nám říká že dekódovaný znak je „a“. V dalším kroku vypočítáme novou hodnotu výstupního čísla. Příklad výpočtu pomocí (2.6):

$$výstupní\ číslo = (0,392578125 - 0)/0,5$$

$$výstupní\ číslo = 0,78515625$$

Nové výstupní číslo je 0,78515625 a leží v intervalu $[0,5; 0,8\overline{33})$, což nám říká, že výsledný znak je „b“. Opakujeme výpočty, dokud nebudeme mít dekódovaný celý řetězec. Po dekódování celého řetězce, pak dostaneme výsledný řetězec, který je „abcaab“ (viz tab. 2.8).

Tab. 2.8: Tabulka určení jednotlivých znaků

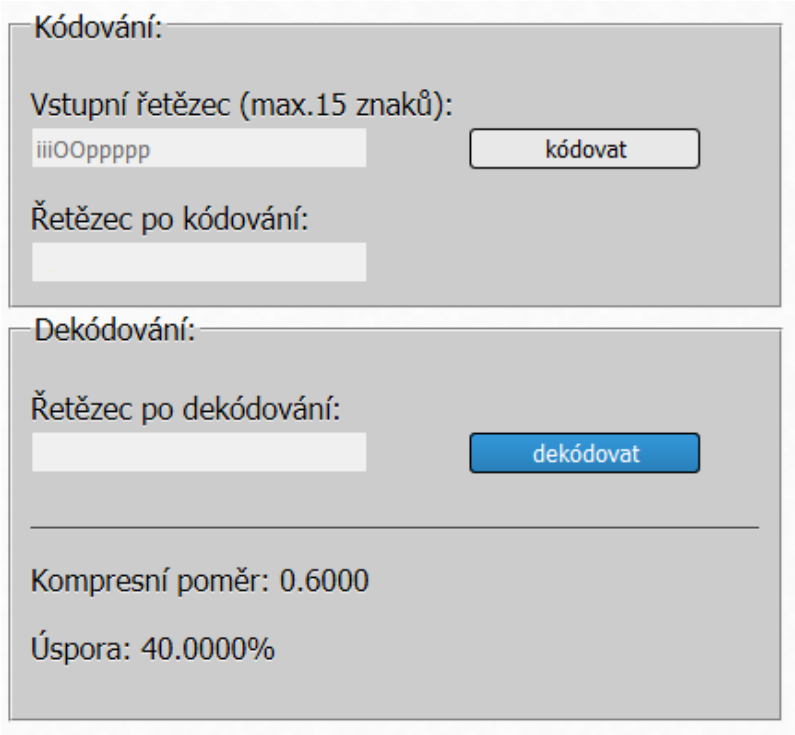
Výstupní číslo	**Low**	**High**	**Range**	**Znak**
0,392578125	0	0,5	0,5	a
0,78515625	0,5	$0,83\overline{3}\overline{3}$	$0,33\overline{3}\overline{3}$	b
0,85546875	$0,83\overline{3}\overline{3}$	1	$0,16\overline{6}\overline{6}$	c
0,1328125	0	0,5	0,5	a
0,265625	0	0,5	0,5	a
0,53125	0,5	$0,83\overline{3}\overline{3}$	$0,33\overline{3}\overline{3}$	b

3 IMPLEMENTACE ALGORITMŮ

Tato část práce se věnuje vývoji jednotlivých aplikací.

3.1 Aplikace – základní RLE

Po načtení aplikace do webového prohlížeče se zobrazí okno aplikace (viz obr. 3.1).



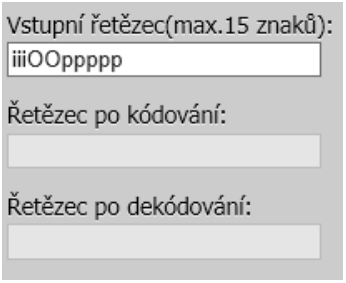
Obr. 3.1: Ukázka okna aplikace RLE

Aplikace umožňuje kódování, dekódování vstupní posloupnosti pomocí algoritmu RLE komprese, a také výpočet kompresního poměru a úspory. Od aplikace se požaduje, aby v jednotlivých krocích dokázala zobrazit hodnoty komprimované posloupnosti a pak získanou komprimovanou posloupnost převést zpátky do výsledné podoby před kompresí dat. Levá horní část aplikace je tvořena třemi textovými poli, přičemž pouze první z nich můžeme editovat a již je předvyplněno (viz obr. 3.2).

V pravé části aplikace se nacházejí dvě tlačítka. Výpočty pro kompresní poměr a úsporu jsou umístěny v levé dolní části aplikace.

3.1.1 Implementace aplikace

Implementace je založena na základní variantě RLE komprese. Jak už bylo zmíněno tato základní varianta kóduje posloupnost do dvojic (počet opakování, znak).



Obr. 3.2: Ukázka textových polí aplikace RLE

Implementace této aplikace provádí RLE kódování pouze s textem, který obsahuje písmena a symboly, a nikoliv čísla. Algoritmus kódování základní RLE komprese je realizován následujícím způsobem:

- algoritmus prochází vstupním řetězcem a počítá počet opakování prvního znaku,
- když narazí na jiný znak, zapíše počet opakování a znak do výstupního řetězce,
- pokud algoritmus narazí na konec řetězce – algoritmus je ukončen,
- pak se do výstupního řetězce zapíše dvojice hodnot – počet opakování a znak výsledné posloupnosti.

Algoritmus dekódování základní RLE komprese je realizován následujícím způsobem:

- algoritmus čte první znak, je-li prvním znakem je číslo, dekodér ví že se jedná o počet opakování znaku a uloží toto číslo,
- algoritmus čte další znak, je-li další znak je písmeno, pak do výstupního řetězce zapíše tento znak , tolikrát, kolik zjistil z předchozího kroků,
- algoritmus pokračuje dál dokud nebude dekódovaný celý vstupní řetězec.

3.1.2 Vytvořené funkce

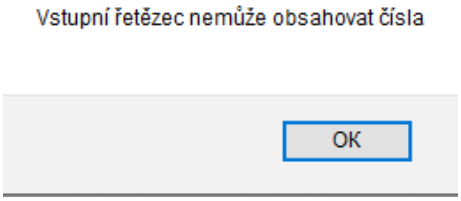
Pro realizaci algoritmu byly vytvořeny tři funkce v jazyku JavaScript:

- `function encode()` – funkce, která kóduje vstupní řetězec a provádí výpočty kompresního poměru a úspory,
- `function decode()` – funkce, která provádí dekódování dat po kompresi,
- `function validate()` – funkce, která omezují vkládání čísel do vstupního řetězce.

3.1.3 Ovládání

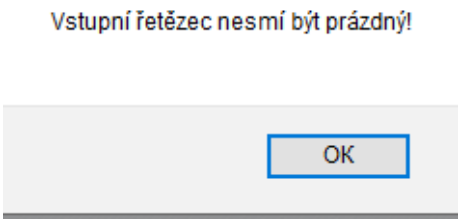
Posloupnost dat, kterou chceme zakódovat se vkládá do textového pole s názvem „Vstupní řetězec“. Do textového pole můžeme vkládat pouze písmena a symboly, zdůvodnění proč – je na straně 20. Jestli se pokusíme vložit do textového pole

číslo, pak aplikace nahlásí chybu (viz obr. 3.3). Také pokud ponecháme textové pole



Obr. 3.3: Ukázka okna s hlášením chyby

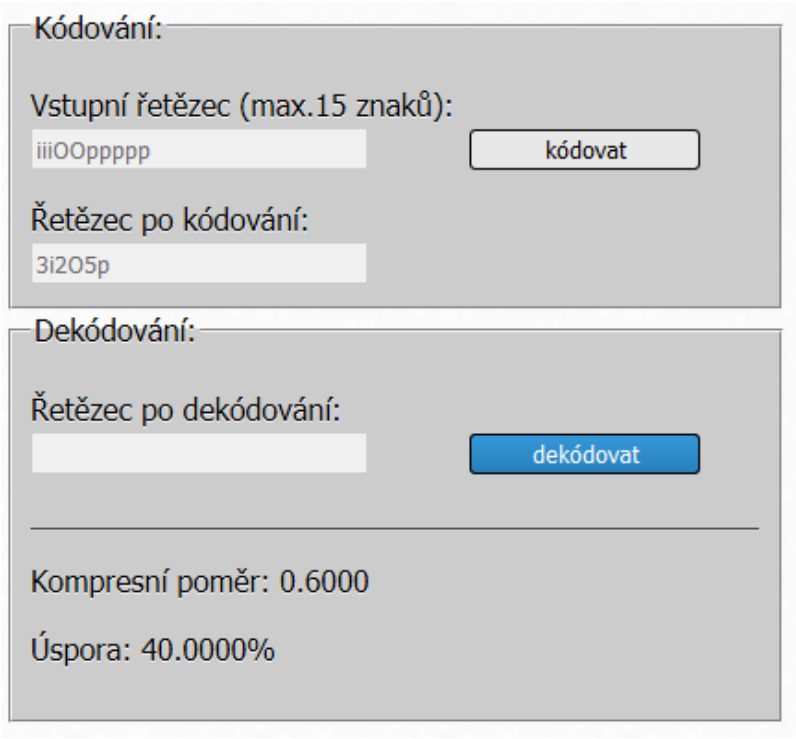
prázdné a stiskneme tlačítko „kódovat“, dojde k chybě (viz obr. 3.15).



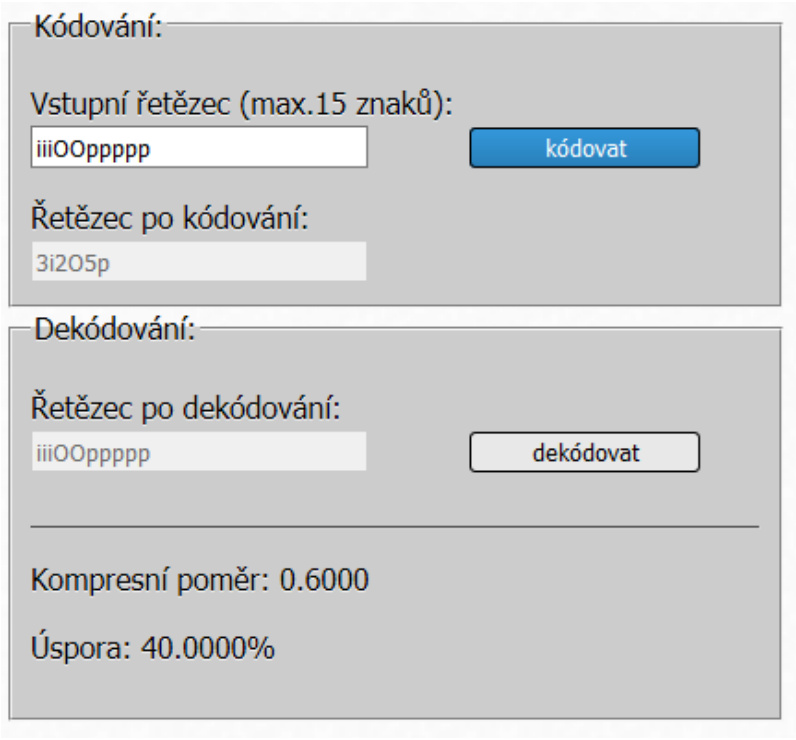
Obr. 3.4: Ukázka okna s hlášením chyby při prázdnem textovém poli

Pokud textové pole obsahuje nějakou posloupnost dat, pak po stisku tlačítka „kódovat“ se spustí funkce kódování a zakódovaná posloupnost se zobrazí v dalším textovém poli s názvem „Řetězec po kódování“, a kromě toho se spočítají kompresní poměr a úspora místa (viz obr. 3.5) podle (2.1) a (2.2).

Posledním krokem můžeme zakódovanou posloupnost dekódovat zpátky do originálního stavu, a to pomocí tlačítka „dekódovat“. Do textového pole se zapíše dekódovaná posloupnost (viz obr. 3.6), která je stejná jako před kódováním.



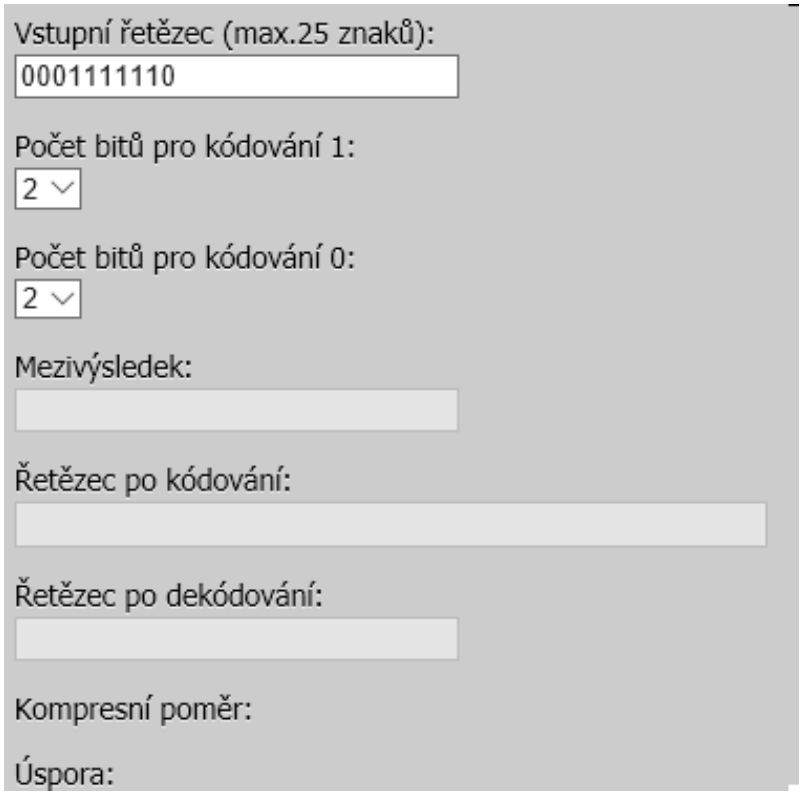
Obr. 3.5: Ukázka zakódované posloupnosti



Obr. 3.6: Ukázka dekódované posloupnosti

3.2 Aplikace – binární RLE

Po načtení aplikace do webového prohlížeče se zobrazí okno aplikace (viz obr. 3.8). Aplikace provádí kódování, dekódování vstupní binární posloupnosti, a také výpočet kompresního poměru a úspory. Od aplikace se očekává, aby dokázala provést kódování vstupní posloupnosti a pak získanou posloupnost převést zpátky do výsledné podoby před kompresí dat. Aplikace se dělí na dvě části. Levá část aplikace je tvořena čtyřmi textovými poli a dvěma prvky, které umožňují výběr počtu bitů pro kódování 1 a 0 (viz obr. 3.7).



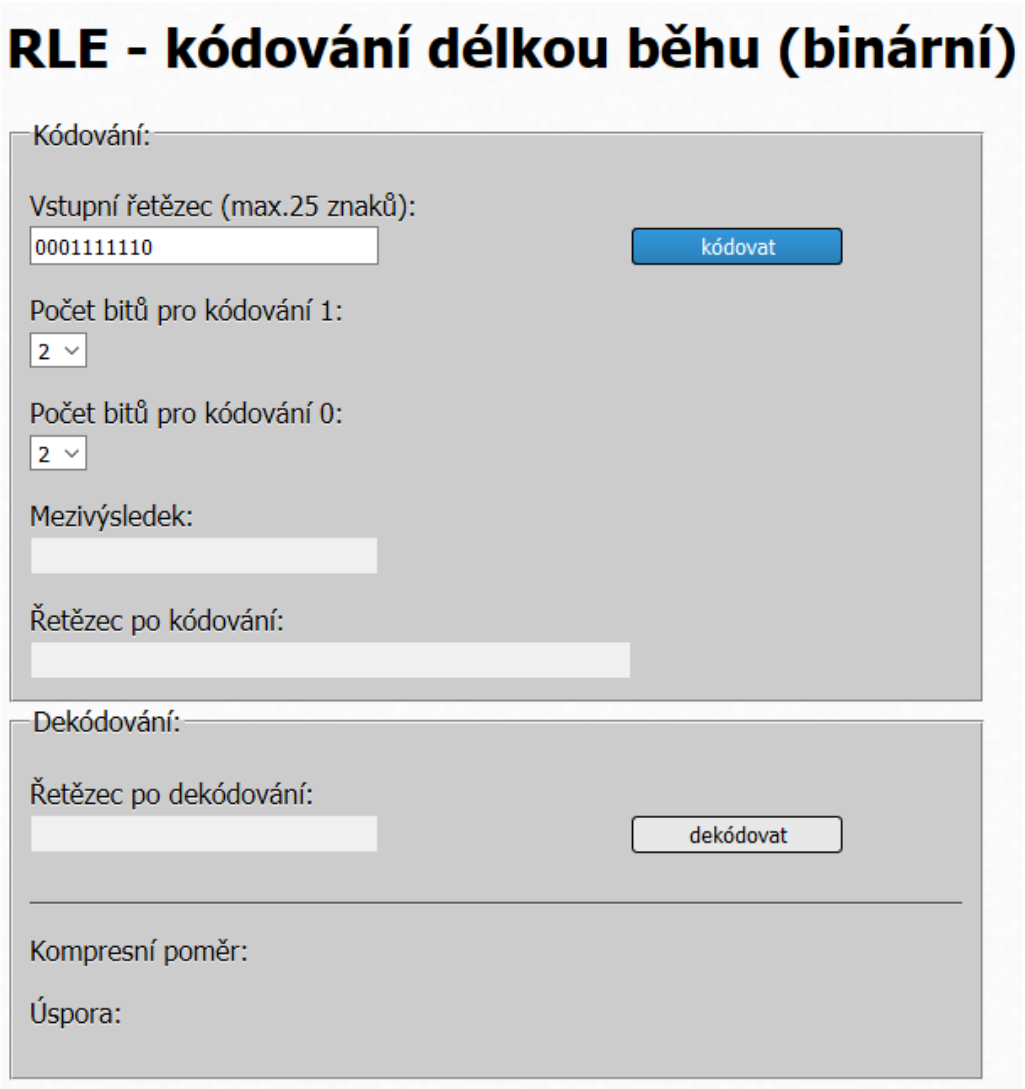
Obr. 3.7: Ukázka textových polí

Pravou část tvoří dvě tlačítka „kódovat“ a „dekódovat“. Výpočty pro kompresní poměr a úsporu jsou umístěny v levé dolní častí aplikace.

3.2.1 Implementace aplikace

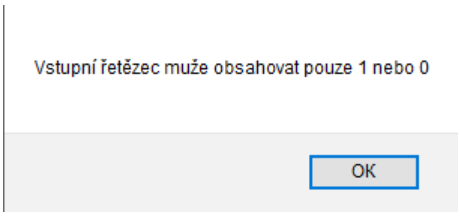
Implementace této aplikace je založena na variantě binární RLE komprese. Algoritmus je realizován následujícím způsobem, kdy se při kódování střídají 0 a 1 [10]:

- za prvé pro kódování 0 a 1 vybereme počet bitů (2,3,4). Například zvolili jsme 2 bity pro kódování, jak 1 tak i 0; To znamená, že maximální počet se opakujících po sobě 0 a 1 může být 3 (11 v binární podobě),



Obr. 3.8: Ukázka okna aplikace binární RLE

- algoritmus prochází vstupním řetězcem, a začíná kódovat posloupnost od 0, to znamená, že jestli první znak vstupního řetězce je 0, zapíše do mezivýsledku počet opakování 0, přičemž maximální počet opakování může být 3 při zvolených 2 bitech.
- jestli první znak není nula, zapíše do mezivýsledku 0, a začne kódovat další znak, který je 1 a zapíše počet opakovaní té 1, pokud další znak je znovu 0, pak do mezivýsledku zapíše počet opakovaní jedničky jako 0, a jde kódovat další 0,
- takovým způsobem pomocí střídaní 0 a 1 zakóduje celý vstupní řetězec,
- pokud algoritmus narazí na konec řetězce – algoritmus je ukončen,
- nakonec algoritmus převede hodnoty mezivýsledku do binární podoby a zapíše výslednou kódovanou posloupnost.



Obr. 3.9: Ukázka hlášení chyby při vložení špatné hodnoty

Algoritmus dekódování binární RLE komprese je realizován následujícím způsobem:

- Dekódování se provádí opačným směrem.

3.2.2 Vytvořené funkce

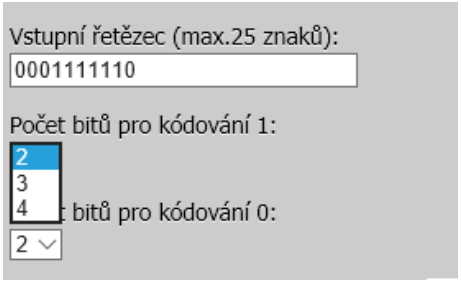
- `function bin_encode()` – funkce, která kóduje vstupní řetězec a provádí výpočty kompresního poměru a úspory,
- `function bin_decode()` – funkce, která provádí dekódování dat po kompresi,
- `function validate()` – funkce, která povoluje vkládat do vstupního řetězce pouze 1 a 0.

3.2.3 Ovládání

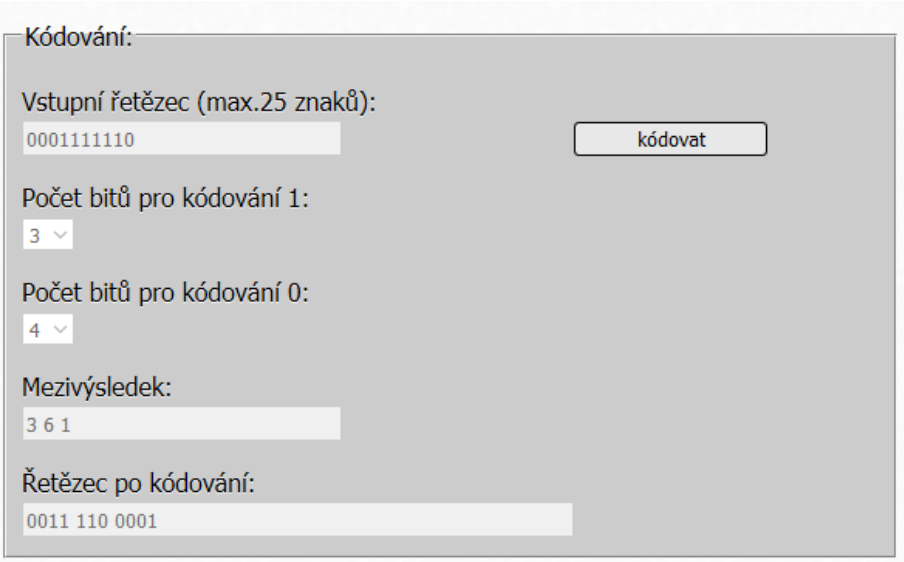
Do vstupního řetězce můžeme vkládat pouze 1 nebo 0. Při pokusu vložit do vstupního řetězce písmeno nebo nějaký symbol, nás aplikace upozorní následujícím hlášením viz obr. 3.9. Stejně při nevyplnění vstupního řetězce obdržíme chybové hlášení identické jako u první aplikace.

Dalším krokem můžeme vybrat kolik bitů budeme používat pro kódování 0 a 1, to se dá nastavit pomocí prvků výběrů (viz obr. 3.10).

Po stisku tlačítka „kódovat“, se do pole s názvem mezivýsledek zapíše počet opakovaní 0 a 1 a do pole s názvem „Řetězec po kódování“ zapíše mezivýsledek převedený do binární podoby (viz obr. 3.11), a také provede jednoduché výpočty kompresního poměru a úspory. Poslední krok, který můžeme provést – je dekódování. Tlačítkem „dekódovat“ spustíme funkci, která provede dekódování a výslednou posloupnost zapíše do pole s názvem „Řetězec po dekódování:“.



Obr. 3.10: Ukázka prvků výběru



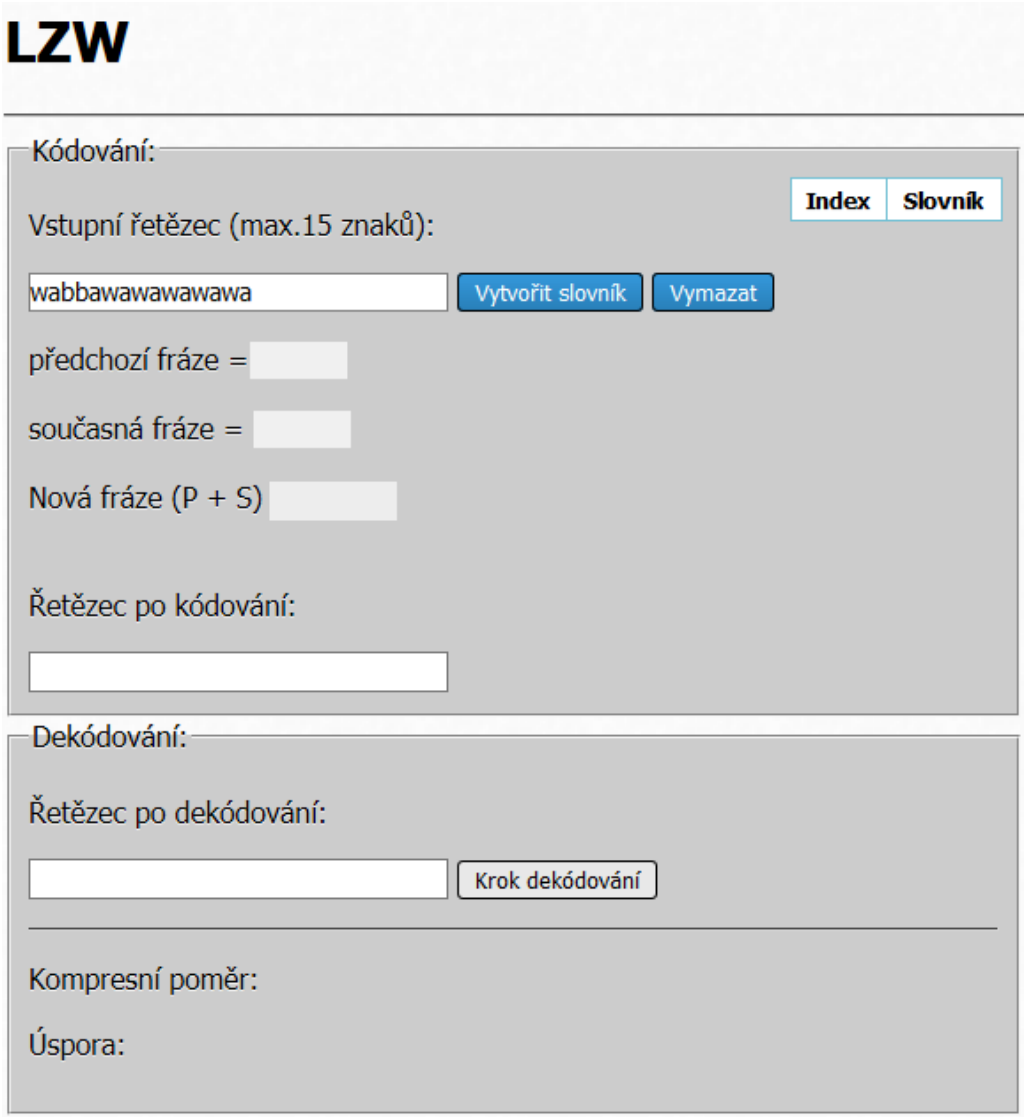
Obr. 3.11: Ukázka kódování

3.3 Aplikace – LZW kódování

Aplikace LZW kódování demonstruje kódování, dekódování vstupní posupnosti takzvanou slovníkovou metodu. Aplikace je tvořena ze dvou částí „Kódování“ a „Dekódování“ (viz obr. 3.12). Levá část aplikace obsahuje šest textových polí, přičemž pouze první z nich můžeme editovat a již je předvyplněno (viz obr. 3.13). Nezbytnou části aplikace jsou tři tlačítka „Vytvořit slovník“, „Krok dekódování“ a „Vymazat“. Po načtení aplikace do webového prohlížeče jsou aktivní pouze tlačítka „Vytvořit slovník“ a „Vymazat“. Aplikaci také tvoří tabulka, která obsahuje slovník s frázemi. Výpočty pro kompresní poměr a úsporu jsou umístěny v dolní části aplikace.

3.3.1 Implementace aplikace

Algoritmus kódování je realizován následujícím způsobem:

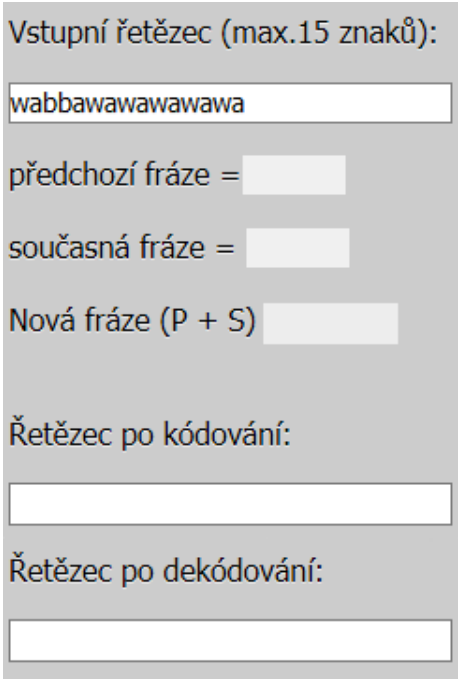


Obr. 3.12: Ukázka okna aplikace LZW

- v prvním kroku se vytváří slovník s jednoznakovými frázemi vstupní posloupnosti, těmto frázím jsou pak přiděleny indexy,
- algoritmus čte první znak vstupní posloupnosti a kontroluje je-li ve slovníku daná fráze. Jestli daná fráze už je ve slovníku, algoritmus čte další znak,
- jestli fráze ještě není ve slovníku, pak tato nová fráze, která je tvořena z předchozí fráze a současné fráze, se zapíše do slovníku a do výstupu se zapíše index předchozí fráze,
- algoritmus pokračuje dál dokud nebude zakódovaný celý vstupní řetězec.

Algoritmus dekódování je realizován následujícím způsobem:

- v prvním kroku inicializuje slovník s jednoznakovými frázemi vstupní posloupnosti,



Obr. 3.13: Ukázka textových polí

- algoritmus čte index zakódované posloupnosti, pak do výstupu zapisuje fráze odpovídající indexu ve slovníku.
- do slovníku se přidává nová fráze, která se tvoří z fráze předchozí a plus první znak fráze současné.
- algoritmus pokračuje dál dokud nebude dekódovaný celý řetězec.

3.3.2 Vytvořené funkce

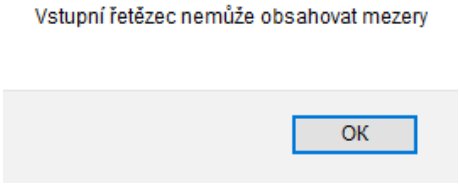
Několik hlavních funkcí, které byly vytvořeny pro implementaci algoritmu LZW kódovaní:

- `function lzw_encode()` – funkce, která kóduje vstupní řetězec a provádí výpočty kompresního poměru a úspory,
- `function lzw_slovnik()` – funkce, která vytvoří slovník,
- `function lzw_decode()` – funkce, která provádí dekódování,
- `function validate()` – funkce, která omezuje vložení mezer do vstupního řetězce.

3.3.3 Ovládání

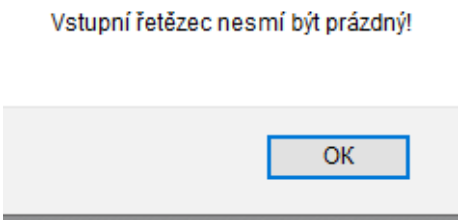
Do textového pole s názvem „Vstupní řetězec“, které po načtení aplikace do webového prohlížeče je už předvyplněno, můžeme vkládat text s maximální délkou 15 znaků. Vstupní text může obsahovat cokoliv kromě mezer. Při pokusu vložit do

vstupního řetězce mezeru aplikace nahlásí chybu(viz obr. 3.14).



Obr. 3.14: Ukázka okna s hlášením chyby při pokusu vložení mezery do textového pole

Aplikace také nahlásí chybu, jestli ponecháme textové pole prázdné a stiskneme tlačítko „Vytvořit slovník“ (viz obr. 3.15).

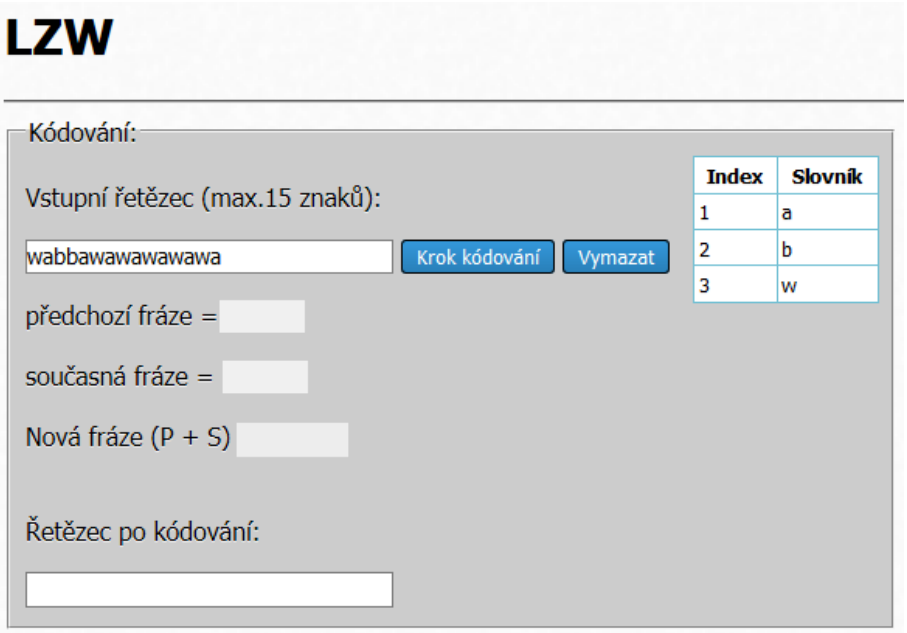


Obr. 3.15: Ukázka okna s hlášením chyby při prázdnem textovém poli

Po stisku tlačítka „Vytvořit slovník“, aplikace vytvoří slovník s jednoznakovými frázemi a každé frázi přidělí index (viz obr. 3.16), pak se název tlačítka změní na „Krok kódování“ a skončí možnost upravovat vstupní řetězec.

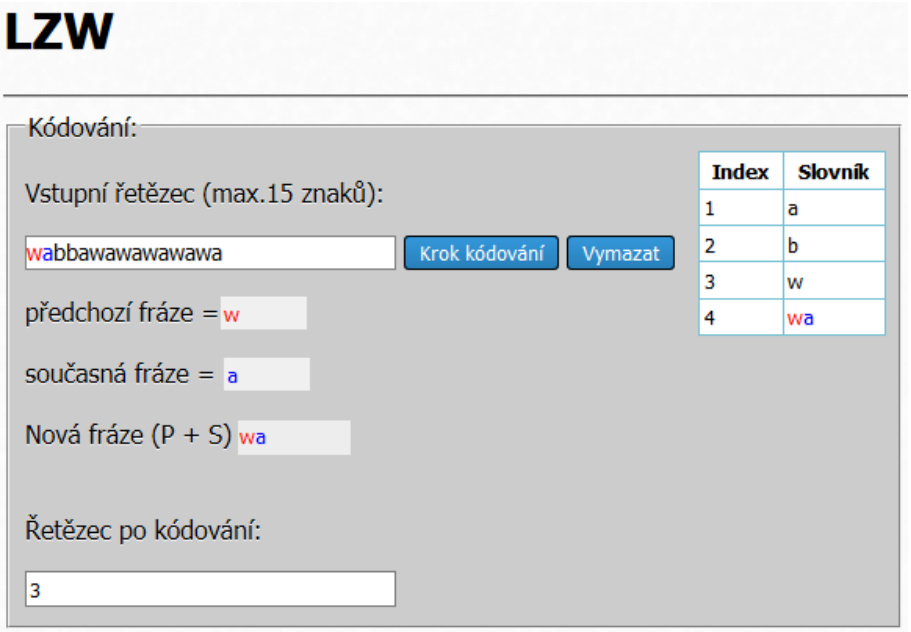
Stiskem tlačítka „Krok kódování“ můžeme zakódovat vstupní text krok za krokem. Pro lepší pochopení fungovaní algoritmu je každá fráze zvýrazněna barvou. Předchozí fráze je zvýrazněna červenou barvou a současná fráze je zvýrazněna modrou barvou. Během kódování do textových políček „předchozí fráze“, „současná fráze“ a „nová fráze“ se zapisují fráze kódované posloupnosti. Do tabulky „Slovník“ se zapisuje nová fráze, která je tvořena z fráze předchozí a současné fráze. Do textového pole „Řetězec po kódování“ se vypisují indexy frází. Obrázek 3.17 demonstruje proces kódování.

Až bude kódování ukončeno dostaneme hlášení, že kódování je ukončeno (viz obr. 3.18). Název tlačítka se změní na „Kódovat znovu“, proběhne výpočet kompresního poměru a úspory místa, a aktivuje se tlačítko „Krok dekódování“. Teď můžeme vybrat jestli chceme zopakovat proces kódování nebo začneme proces dekódování. Třetí možnost je restartovat aplikaci stiskem tlačítka „Vymazat“. Stiskem tlačítka „Krok dekódování“ spustíme proces dekódování. Po dekódování se název tlačítka

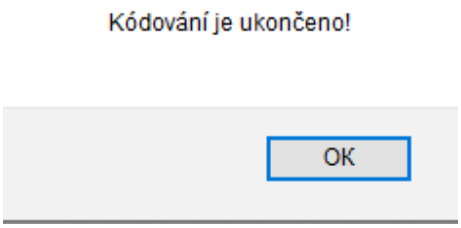


Obr. 3.16: Ukázka okna vytvoření slovníku s jednoznakovými frázemi

změní na „Dekódovat znovu“ a můžeme vybrat jestli chceme zapakovat proces dekódování nebo začneme proces kódování nové vstupní posloupnosti. Stiskem tlačítka „Vymazat“ můžeme restartovat celou aplikaci.



Obr. 3.17: Ukázka procesu kódování



Obr. 3.18: Ukázka okna hlášení ukončení kódování

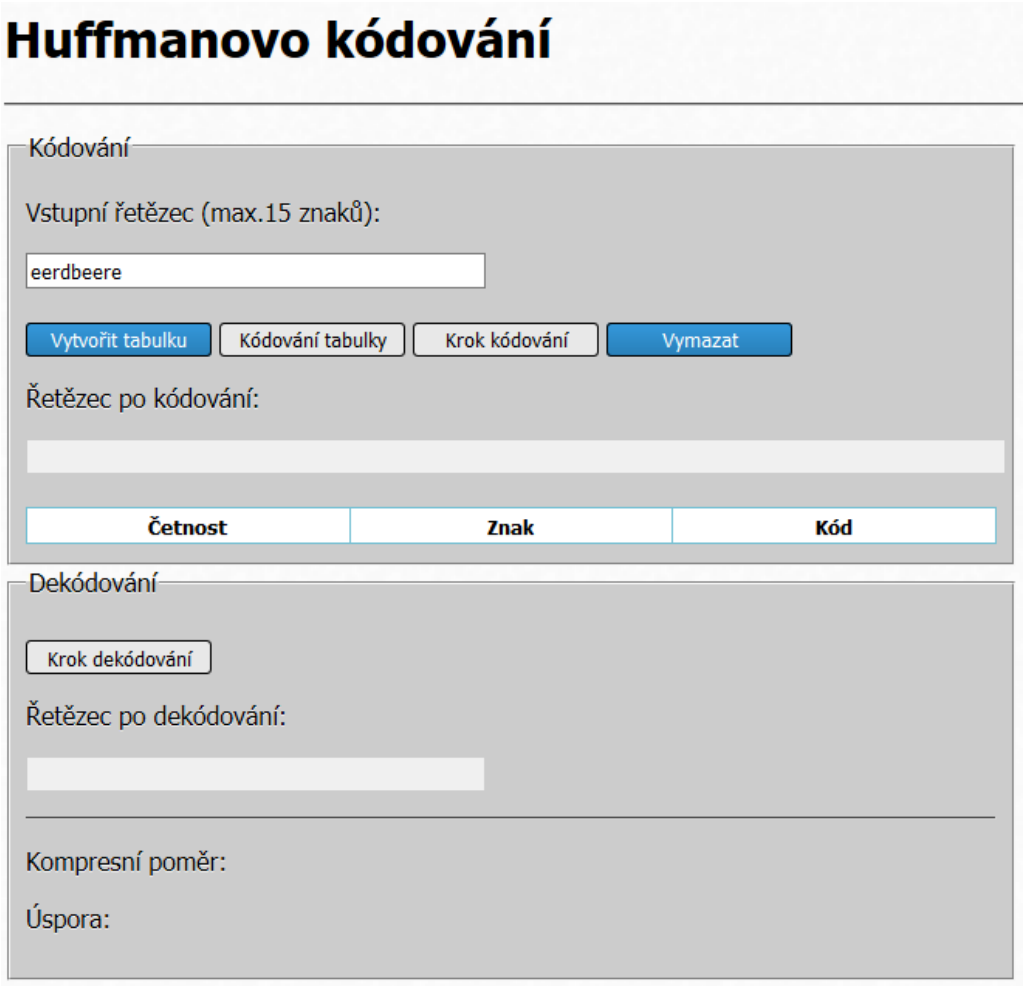
3.4 Aplikace – Huffmanovo kódování

Aplikace demonstruje Huffmanovo kódování, které využívá takzvaného prefixového kódu. Principem prefixového kódu je, že žádné kódové slovo není předponou jiného. Aplikace je dle obrázku 3.19 tvořena ze dvou částí „Kódování“ a „Dekódování“ . Část „Kódování“ je tvořena dvěma poli pro vstupní a výstupní text, tabulkou pro tvorbu kódu a čtyřmi tlačítky. Část „Dekódování“ je tvořena jedním polem a tlačítkem. Výpočty pro kompresní poměr a úsporu jsou umístěny v dolní částí aplikace.

3.4.1 Implementace aplikace

Algoritmus kódování je realizován následujícím způsobem:

- v prvním kroku algoritmus projde vstupním řetězcem a vytvoří tabulku četnosti výskytu jednotlivých znaků,



Obr. 3.19: Ukázka okna aplikace Huffmanovo kódování

- podle četnosti výskytu znaků vytvoříme frontu, kde priorita bude podle četnosti výskytu jednotlivých znaků,
- začne tvořit uzly binárního stromu. Dva první znaky z fronty sloučí a z nich vytvoří nový uzel stromu, který přidá do fronty. Četnost nového uzlu je součet četností sečtených prvků,
- opakujeme předchozí kroky tak dlouho, dokud se nevytvoří celý strom,
- zakódujeme strom 1 a 0. Uzlům, které se nachází v levé části stromu přidává 0 a uzlům, které se nachází v pravé části stromu přidává 1,
- v dalším kroku provede kódování vstupního řetězce do binárního tvaru podle binárního stromu, který byl vytvořen v předchozím kroku,
- kódování se provádí tak dlouho, dokud nebude zakódován celý vstupní řetězec.

Algoritmus dekódování je realizován následujícím způsobem:

- dekódování se provádí pomocí binárního stromu,
- dekodér projde stromem od kořene k uzlům a zjistí, kterému znaku odpovídá

kód,

- dekódování se provádí tak dlouho, dokud nebude dekódován celý řetězec.

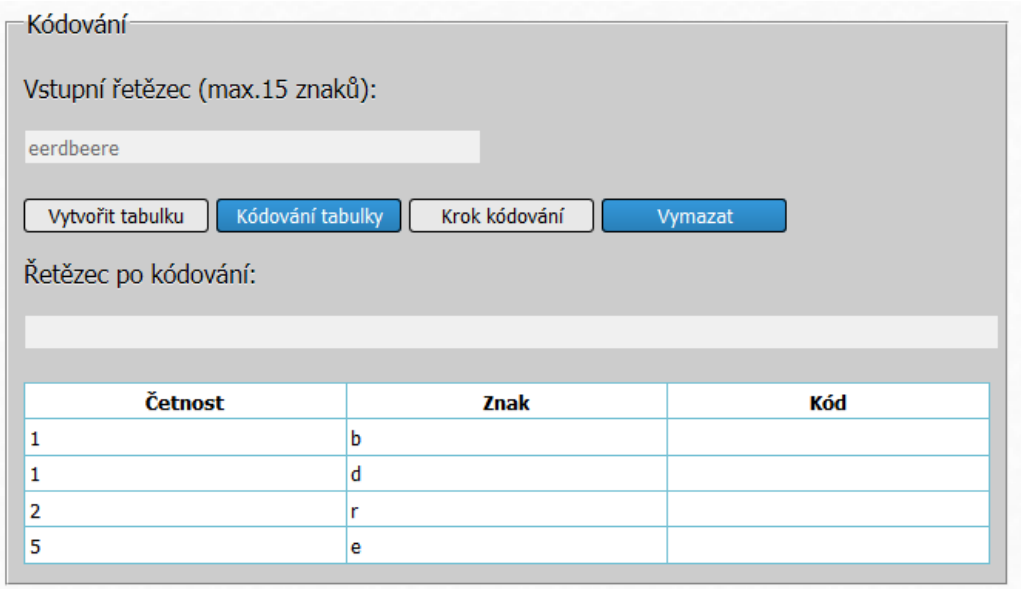
3.4.2 Vytvořené funkce

Několik hlavních funkcí, které byly vytvořeny pro implementaci algoritmu Huffmanovo kódování:

- `function cetnost()` – provádí výpočet četnosti jednotlivých znaků,
- `function slovnik()` – vytváří tabulku četnosti,
- `function strom0()` – vytváří strom uzlů,
- `function nastavkod()` – přidává uzlům kódy 0 nebo 1,
- `function encodeClick()` – zakóduje vstupní posloupnost do binárního tvaru,
- `function decodeClick` – provádí dekódování zakódované posloupnosti,

3.4.3 Ovládání

Do textového pole „Vstupní řetězec“ zapíšeme text, který potřebujeme zakódovat. Maximální počet znaku vstupního řetězce je 15 znaků. Stiskem tlačítka „Vytvořit tabulku“ vytvoříme tabulku četnosti (viz obr. 3.20). Následně klikáním na tlačítko

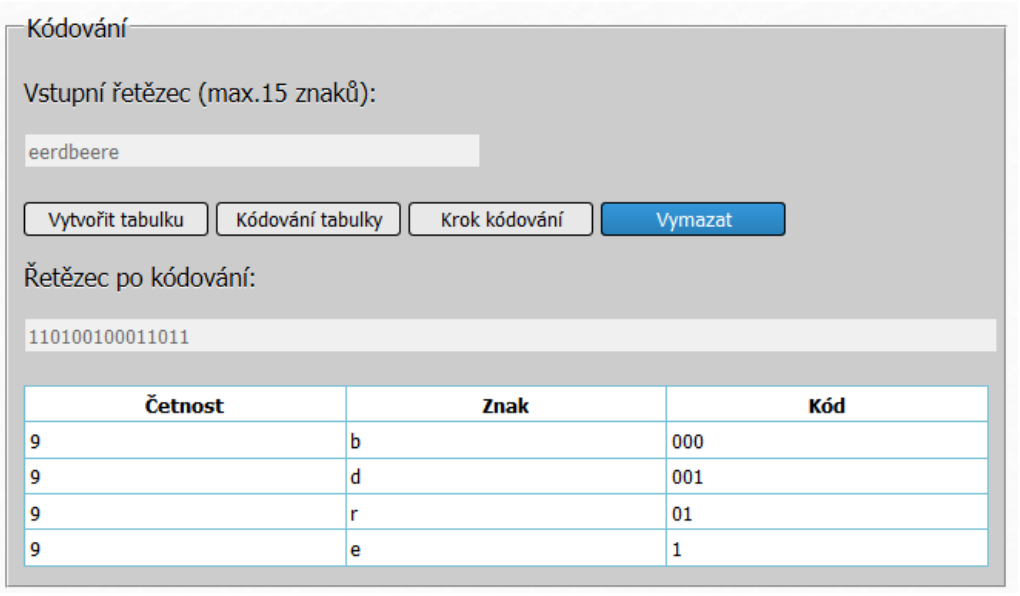


Obr. 3.20: Ukázka okna vytvoření tabulky četnosti

„Kódování tabulky“ provedeme krok po kroku slučování prvků nejmenších četností a přidělování kódů. V tabulce jsou aktuálně kódované znaky zvýrazněny červenou barvou (viz obr. 3.21).

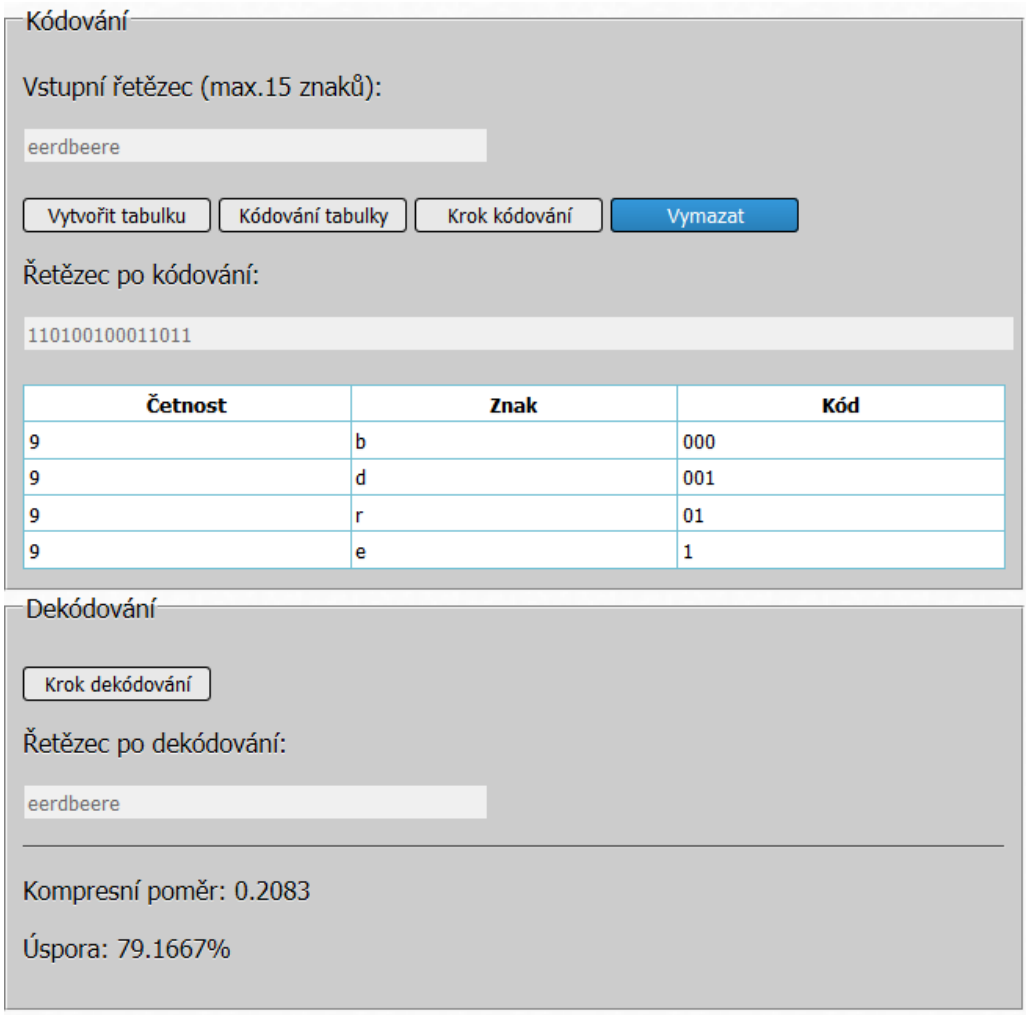
Obr. 3.21: Ukázka kódování tabulky

Poté až bude tabulka zakódovaná, stiskem tlačítka „Krok kódování“ spustím kódování vstupního řetězce do binárního tvaru. Jednotlivým znakům se přidává příslušný kód z tabulky a probíhá výpočet kompresního poměru a úspory místa. Zakódovaná posloupnost se vypisuje v textovém poli „Řetězec po kódování“ (viz obr. 3.22). Dekódování se provádí tlačítkem „Krok dekódovaní“ a dekódovaná po-



Obr. 3.22: Ukázka kódování jednotlivých znaků

sloupnost se vypisuje v textovém poli „Řetězec po dekódování“ (viz obr. 3.23). Smazání hodnot v tabulce se provádí pomocí tlačítka „Vymazat“.



Obr. 3.23: Ukázka dekódování jednotlivých znaků

3.5 Aplikace – Aritmetické kódování

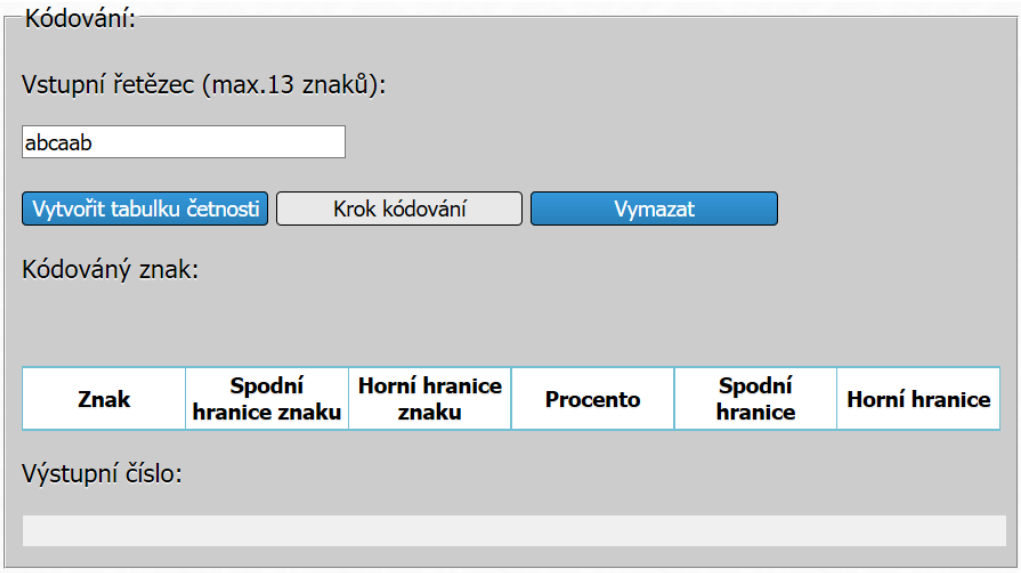
Pomocí této aplikace je možné vyzkoušet Aritmetické kódování, kde se provádí kódování celého vstupního řetězce do jednoho čísla. Aplikaci tvoří dvě částí „Kódování“(viz obr. 3.24) a „Dekódování“(viz obr. 3.25).

Část „Kódování“ je tvořena dvěma poli pro vstupní a výstupní text, tabulkou pro kódování vstupního řetězce a třemi tlačítky. Část „Dekódování“ je tvořena jedním polem, tlačítkem a tabulkou, ve které zobrazuje proces dekódování. Výpočty pro kompresní poměr a úsporu jsou umístěny v dolní části aplikace.

3.5.1 Implementace aplikace

Algoritmus kódování je realizován následujícím způsobem:

- nastaví počáteční interval na $[0, 1)$,

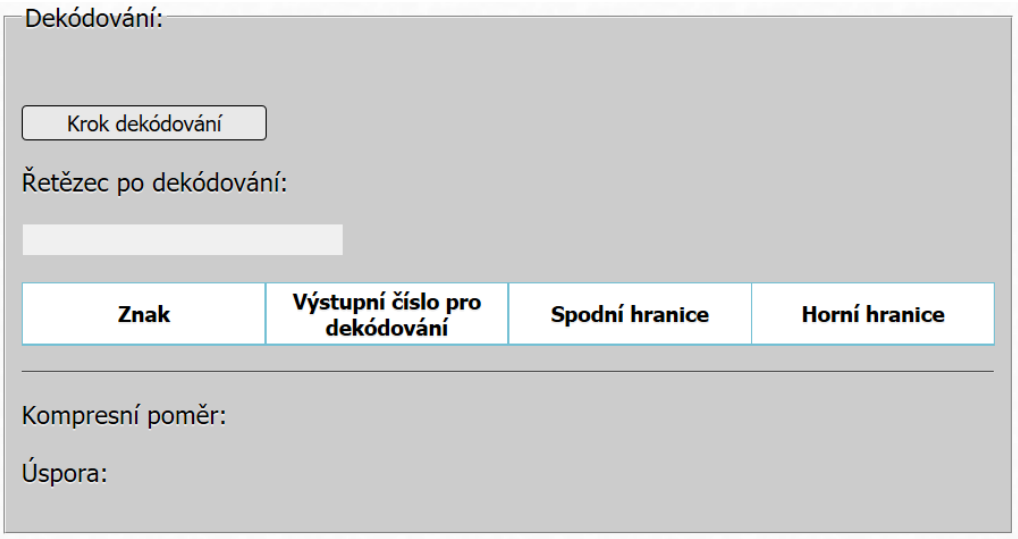


Obr. 3.24: Ukázka části „Kódování“ okna aplikace Aritmetické kódování

- algoritmus projde vstupním řetězcem a spočítá četnosti výskytu jednotlivých znaků,
- provede rozdělení pravděpodobnosti výskytu znaků do intervalu $[0, 1)$,
- provede výpočet nových hranic intervalu pro každý znak vstupního řetězce pomocí (2.3) a (2.4),
- provede výpočet výstupního čísla, které je střední hodnota z posledního intervalu,
- provede převod výstupního čísla do binárního tvaru, potřebný počet bitů pro zakódovaní vstupní posloupnosti zjistí podle (2.5),
- kódování je ukončeno.

Algoritmus dekódování je realizován následujícím způsobem:

- provádí převod výstupního čísla z binárního tvaru do desítkového tvaru,
- nastaví počáteční interval na $[0, 1)$,
- rozdělí interval $[0, 1)$ podle pravděpodobností výskytu jednotlivých znaků,
- zkontroluje do jakého intervalu spadá výstupní číslo a tím určí výsledný znak,
- provede výpočet nové hodnoty výstupního čísla pomocí (2.6),
- zkontroluje do jakého intervalu spadá nové výstupní číslo a tím určí výsledný znak,
- poslední dva kroky se opakují tak dlouho, dokud nebude dekódován celý řetězec.



Obr. 3.25: Ukázka části „Dekódování“ okna aplikace Aritmetické kódování

3.5.2 Vytvořené funkce

Několik hlavních funkcí, které byly vytvořeny pro implementaci algoritmu Aritmetické kódování:

- `function vypCetnost()` – provádí výpočet četnostní jednotlivých znaků,
- `function encodeClick()` – provádí kódování vstupní posloupnosti,
- `function toBin()` – provádí převod výstupního čísla do binárního tvaru,
- `function toDes()` – provádí převod výstupního čísla z binárního tvaru,
- `function decodeClick()` – provádí dekódování zakódované posloupnosti.

3.5.3 Ovládání

Vstupní text, který chceme zakódovat se vkládá do textového pole s názvem „Vstupní řetězec“. Maximální počet znaků vstupního řetězce je 13 znaků. Stiskem tlačítka „Vytvořit tabulku četnosti“ spočítá algoritmus četnosti jednotlivých znaků vstupní posloupnosti a rozdělí je do intervalu $[0, 1)$ (viz obr. 3.26). Tabulka obsahuje informace o jednotlivých znacích, jejich četnosti, spodní a horní hranici.

Následně klikáním na tlačítko „Kódování tabulky“ provedeme krok po kroku výpočet nových hranic intervalu. V průběhu kódování je aktuálně kódovaný znak zvýrazněn červenou barvou (viz obr. 3.27).

V posledním kroku kódování se vypočítá výstupní číslo a zároveň se převádí do binárního tvaru. Výstupní číslo se zapisuje do textového pole „Výstupní číslo“ (viz obr. 3.28) a probíhá výpočet kompresního poměru a úspory místa.

Znak	Spodní hranice znaku	Horní hranice znaku	Procento
a	0	0.5	50
b	0.5	0.8333333333333333	33.33333333333333
c	0.8333333333333333	1	16.666666666666664

Obr. 3.26: Ukázka okna vytvoření tabulky četnosti

Kódovaný znak:

abcaab

Znak	Spodní hranice znaku	Horní hranice znaku	Procento	Spodní hranice	Horní hranice
a	0	0.5	50	0	0.5
b	0.5	0.8333333333333333	33.33333333333333		
c	0.8333333333333333	1	16.666666666666664		

Obr. 3.27: Ukázka kódování jednotlivých znaků

Výstupní číslo:

0.3935185185185185

0110010010

Obr. 3.28: Ukázka výstupního čísla

Posledním krokem můžeme provést dekódování zakódované posloupnosti zpátky do originálního stavu, a to pomocí tlačítka „Krok dekódování“. Po stisku tlačítka „Krok dekódování“ bude výstupní číslo v binárním tvaru znovu převedeno do tvaru desítkového. Výstupní číslo v desítkovém tvaru se vypíše do textového pole (viz obr. 3.29) a zároveň se v tabulce vypiše dekódovaný znak, a spodní a horní hranice intervalu, ve kterém leží tento znak.

Následně se vypočítá nové výstupní číslo. Dekódování se provádí dokud nebude dekódován celý řetězec. Do textového pole „Řetězec po dekódování“ se zapíše dekódovaná posloupnost (viz obr. 3.30), která je stejná jako před kódováním. Smazání hodnot v tabulkách se provádí pomocí tlačítka „Vymazat“.

0110010010=0.392578125

Krok dekódování

Řetězec po dekódování:

a

Znak	Výstupní číslo pro dekódování	Spodní hranice	Horní hranice
a	0.392578125	0	0.5

Obr. 3.29: Ukázka dekódování jednotlivých znaků

Dekódování:

0110010010=0.392578125

Krok dekódování

Řetězec po dekódování:

abcaab

Znak	Výstupní číslo pro dekódování	Spodní hranice	Horní hranice
a	0.392578125	0	0.5
b	0.78515625	0.5	0.8333333333333333
c	0.8554687500000002	0.8333333333333333	1
a	0.13281250000000172	0	0.5
a	0.26562500000000344	0	0.5
b	0.5312500000000069	0.5	0.8333333333333333

Kompresní poměr: 0.2083

Úspora: 79.1667%

Obr. 3.30: Ukázka dekódované posloupnosti

4 ZÁVĚR

Cílem této práce bylo nastudovat teorie z oblasti bezeztrátových kompresních algoritmů, a to Huffmanovo kódování, LZW, RLE a aritmetické kódovaní. Navrhnout uživatelské rozhraní pro každý z algoritmů a implementovat pomocí jazyků HTML5 a JavaScript.

V první části práce jsem popsal teorie, které se tykají webových technologií. Základně jsem popsal jazyky HTML5, JavaScript a CSS.

Ve druhé části byly popsány algoritmy bezeztrátové komprese dat. Jsou tady popsány procesy kódování a dekódování jednotlivých algoritmů.

V praktické části této práce jsem vytvořil pět aplikací za pomoci jazyků JavaScript a HTML5. Pro tvorbu uživatelského rozhraní jsem používal HTML5 kvůli jeho aktuálnosti a jednoduché implementaci. Při psaní kódu byl využit jako vývojové prostředí textový editor Sublime Text 3. V praktické části jsou popsány vytvořené funkce, ovládání a implementace aplikací.

První aplikace „základní RLE“ názorně demonstruje kódování a dekódování textové posloupnosti.

Druhá aplikace „binární RLE“ nám demonstruje kódování a dekódování posloupnosti v binární podobě. Navíc umožňuje pro kódování 0 a 1 vybrat počet bitů.

Třetí aplikace provádí kódování a dekódování textu pomocí algoritmu Huffmanova kódování, které využívá takzvaného prefixového kódu.

Čtvrtá aplikace „LZW kódování“ demonstruje kódování a dekódování textu takzvanou slovníkovou metodu.

Pátá aplikace „Aritmetické kódování“ demonstruje fungovaní algoritmu aritmetického kódování, kde se provádí kódování celého vstupního řetězce do jednoho čísla. Aplikace umožňuje provádět aritmetické kódování a dekódování.

Každá aplikace obsahuje informace ohledně kompresního poměru a úspory místa. Při vypočtu kompresní poměru se předpokládá, že každý nekódovaný znak je popsán osmi bity.

Jednotlivé aplikace byly otestovány v následujících webových prohlížečích Google Chrome, Yandex Browser, Mozilla Firefox, Microsoft Edge. Stanovený cíl práce byl splněn. Vytvořené aplikace jsou plně funkční.

LITERATURA

[1] STAUDEK, J. *Komprese dat* [online]. 4. 04. 2006 [cit. 15. 11. 2015]. Dostupné z URL: <http://www.fi.muni.cz/usr/staudek/komprese/Komprese_dat.pdf>.

[2] BROWN, T.B., BUTTERS, K., PANDA, S. *HTML5 Okamžitě*. Překlad: Ondřej Baše 1. vyd. Brno: Computer Press, 2014. 256 s. ISBN 978-80-251-4296-7.

[3] TASHKOV, P. *Web mastering na 100 %:HTML, CSS, JavaScript, PHP, CMS, AJAX, spuštění*. Sankt Petěrburk: Piter, 2010. 512 s. ISBN 978-5-49807-826-7.

[4] CASTRO, E., HYSLOP, B. *HTML a CSS3: Názorný průvodce tvorbou WWW stránek*. Překlad: Ondřej Baše, Kristýna Baše 1. vyd. Brno: Computer Press, 2012. 439 s. ISBN 978-80-251-3733-8.

[5] ZAKAS, N. *JavaScript pro webové vývojáře: Programujeme profesionálně*. Překlad: Lukáš Krejčí 1. vyd. Brno: Computer Press, 2009. 832 s. ISBN 978-80-251-2509-0.

[6] ZAKAS, N. *Professional JavaScript for Web Developers*. Indianapolis: John Wiley & Sons, Inc., 2012. 919 s. ISBN 978-1-118-02669-4.

[7] THAU, D. *Velký průvodce JavaScriptem: tvorba interaktivních webových stránek*. Překlad: Helena Danihelková 1. vyd. Praha: Grada Publishing, 2009. 520 s. ISBN 978-80-247-2211-5.

[8] CASTRO, E., HYSLOP, B. *HTML5 and CSS3: Visual QuickStart Guide*. 7. vyd. Berkeley: Peachpit Press, 2012. 606 s. ISBN 978-0-321-71961-4.

[9] SALOMON, D. *Data Compression: The Complete Reference*. 4. vyd. London: Springer, 2011. 1092 s. ISBN 978-1-84628-602-5.

[10] PERLMAN, W., SAID, A. *Digital compression: Principles and Practice*. New York: Cambridge University Press, 2011. 419 s. ISBN 978-0-521-89982-6.

[11] VEČERKA, A. *Komprese dat* [online]. Olomouc: Univerzita Palackého 2008. Dostupné z URL: <http://phoenix.inf.upol.cz/esf/ucebni/komprese.pdf/>.

[12] Sublime Text. [online]. 2015 [cit. 07. 12. 2015]. Dostupné z URL: <http://www.sublimetext.com/>.

[13] W3C. *All standards and drafts.* [online]. 2015 [cit. 08. 11. 2015]. Dostupné z URL: <http://www.w3.org/TR/#tr_CSS/>.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

RLE	Run-Length Encoding
HTML	HyperText Markup Language
SGML	Standard Generalized Markup Language
DOM	Document Object Model
API	Application Programming Interface
BOM	Browser Object Model
CSS	Cascading Style Sheets
LZW	Lempel-Ziv-Welch

SEZNAM PŘÍLOH

A OBSAH PŘILOŽENÉHO CD

Na přiloženém CD se nachází elektronická verze této bakalářské práce ve formátu PDF, zdrojové kódy ke všem 5 aplikacím.

Struktura CD je následující:

1. Aplikace – Aritmetické kódování
    - `\aplikace\Aritmetické kódování\Aritmeticke kodovani.html`
    - `\aplikace\Aritmetické kódování\sc.js`
    - `\aplikace\Aritmetické kódování\style.css`
2. Aplikace – Huffmanovo kódování
    - `\aplikace\Huffmanovo kódování\Huffman.html`
    - `\aplikace\Huffmanovo kódování\sc.js`
    - `\aplikace\Huffmanovo kódování\style.css`
3. Aplikace – LZW kódování
    - `\aplikace\LZW kódování\lzw.html`
    - `\aplikace\LZW kódování\sc.js`
    - `\aplikace\LZW kódování\style.css`
4. Aplikace – binární RLE
    - `\aplikace\RLE kódování\RLE_binární.html`
    - `\aplikace\RLE kódování\script.js`
    - `\aplikace\RLE kódování\style.css`
5. Aplikace – základní RLE
    - `\aplikace\RLE kódování\RLE_základní.html`
    - `\aplikace\RLE kódování\script.js`
    - `\aplikace\RLE kódování\style.css`
6. BP.pdf